

# SOFT COMPUTING AND INTELLIGENT SYSTEMS DESIGN

THEORY, TOOLS AND APPLICATIONS

FAKHREDDINE O. KARRY AND CLARENCE DE SILVA



ADDISON  
WESLEY

Additional student support at  
[www.booksites.net/karray](http://www.booksites.net/karray)

## Soft Computing and Intelligent Systems Design



We work with leading authors to develop the strongest educational materials in computing, bringing cutting-edge thinking and best learning practice to a global market.

Under a range of well-known imprints, including Addison Wesley, we craft high quality print and electronic publications which help readers to understand and apply their content, whether studying or at work.

To find out more about the complete range of our publishing, please visit us on the World Wide Web at:  
[www.pearsoned.co.uk](http://www.pearsoned.co.uk)

# Soft Computing and Intelligent Systems Design

Theory, Tools and Applications

Fakhreddine O. Karray and  
Clarence de Silva



Harlow, England • London • New York • Boston • San Francisco • Toronto  
Sydney • Tokyo • Singapore • Hong Kong • Seoul • Taipei • New Delhi  
Cape Town • Madrid • Mexico City • Amsterdam • Munich • Paris • Milan

**Pearson Education Limited**  
Edinburgh Gate  
Harlow  
Essex CM20 2JE  
England

*and Associated Companies throughout the world*

*Visit us on the World Wide Web at:*  
[www.pearsoned.co.uk](http://www.pearsoned.co.uk)

**First published 2004**

© Pearson Education Limited 2004

The rights of Fakhreddine Karray and Clarence de Silva to be identified as authors of this work have been asserted by them in accordance with the Copyright, Designs and Patents Act 1988.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without either the prior written permission of the publisher or a licence permitting restricted copying in the United Kingdom issued by the Copyright Licensing Agency Ltd, 90 Tottenham Court Road, London W1T 4LP.

The programs in this book have been included for their instructional value. They have been tested with care but are not guaranteed for any particular purpose. The publisher does not offer any warranties or representations nor does it accept any liabilities with respect to the programs.

All trademarks used herein are the property of their respective owners. The use of any trademark in this text does not vest in the author or publisher any trademark ownership rights in such trademarks, nor does the use of such trademarks imply any affiliation with or endorsement of this book by such owners.

ISBN 0 321 11617 8

**British Library Cataloguing-in-Publication Data**  
A catalogue record for this book is available from the British Library

**Library of Congress Cataloguing-in-Publication Data**  
Karray, Fakhreddine.

Soft computing and tools of intelligent systems design / Fakhreddine Karray and Clarence de Silva.

p. cm.

Includes bibliographical references and index.

ISBN 0-321-11617-8 (pbk.)

1. Soft computing. 2. Expert systems (Computer science) I. De Silva, Clarence W. II. Title.

QA76.9.S63K37 2004

006.3—dc22

2003063628

10 9 8 7 6 5 4 3 2 1  
09 08 07 06 05 04

Typeset in 10/12pt Sabon by 35

Printed and bound in Great Britain by Biddles Ltd, Guildford and King's Lynn

*The publisher's policy is to use paper manufactured from sustainable forests.*

To  
my mother  
and  
the memory of my father

Fakhreddine Karray

To  
my mother  
and  
my father

Clarence de Silva



# Contents

<i>Preface</i>	xvii
<i>Acknowledgements</i>	xxi
<i>List of acronyms</i>	xxiii

## Part I Fuzzy logic and fuzzy control 1

Chapter 1	Introduction to intelligent systems and soft computing	3
1.1	Introduction	3
1.2	Intelligent systems	3
1.2.1	Machine intelligence	5
1.2.2	Meaning of intelligence	6
1.2.3	Dynamics of intelligence	8
1.2.4	Intelligent machines	9
1.3	Knowledge-based systems	12
1.3.1	Architectures of knowledge-based systems	14
1.3.2	Production systems	14
1.3.2.1	Reasoning strategies	16
1.3.2.2	Conflict resolution methods	17
1.3.3	Frame-based systems	17
1.3.4	Blackboard systems	20
1.3.5	Object-oriented programming	22
1.3.6	Expert systems	22
1.3.6.1	Development of an expert system	24
1.3.6.2	Knowledge engineering	25
1.3.6.3	Applications	25
1.4	Knowledge representation and processing	26
1.4.1	Semantic networks	27
1.4.2	Crisp logic	28
1.4.2.1	Crisp sets	28
1.4.2.2	Operations of sets	29

	1.4.2.3 Logic	30
	1.4.2.4 Correspondence between sets and logic	32
	1.4.2.5 Logic processing (reasoning and inference)	33
	1.4.2.6 Laws of logic	33
	1.4.2.7 Rules of inference	35
	1.4.2.8 Propositional calculus and predicate calculus	37
1.5	Soft computing	38
	1.5.1 Fuzzy logic	38
	1.5.2 Neural networks	41
	1.5.3 Genetic algorithms	43
	1.5.4 Probabilistic reasoning	44
	1.5.5 Approximation and intelligence	45
	1.5.6 Technology needs	50
1.6	Summary	51
	Problems	51
	References	54
Chapter 2	Fundamentals of fuzzy logic systems	57
2.1	Introduction	57
2.2	Background	58
	2.2.1 Evolution of fuzzy logic	60
	2.2.1.1 Popular applications	61
	2.2.2 Stages of development of an intelligent product	63
	2.2.3 Use of fuzzy logic in expert systems	64
2.3	Fuzzy sets	65
	2.3.1 Membership function	66
	2.3.2 Symbolic representation	66
2.4	Fuzzy logic operations	68
	2.4.1 Complement (negation, NOT)	69
	2.4.2 Union (disjunction, OR)	70
	2.4.3 Intersection (conjunction, AND)	72
	2.4.4 Basic laws of fuzzy logic	73
2.5	Generalized fuzzy operations	76
	2.5.1 Generalized fuzzy complement	76
	2.5.2 Triangular norms	77
	2.5.2.1 T-norm (generalized intersection)	77
	2.5.2.2 S-norm or triangular conorm (generalized union)	78
	2.5.3 Set inclusion ( $A \subset B$ )	80
	2.5.3.1 Grade of inclusion	81
	2.5.4 Set equality ( $A = B$ )	82
	2.5.4.1 Grade of equality	82

2.6	Implication (if-then)	82
2.6.1	Considerations of fuzzy implication	83
2.7	Some definitions	89
2.7.1	Height of a fuzzy set	89
2.7.2	Support set	89
2.7.3	$\alpha$ -cut of a fuzzy set	90
2.7.4	Representation theorem	90
2.8	Fuzziness and fuzzy resolution	91
2.8.1	Fuzzy resolution	91
2.8.2	Degree of fuzziness	93
2.8.2.1	Measures of fuzziness	93
2.9	Fuzzy relations	97
2.9.1	Analytical representation of a fuzzy relation	99
2.9.2	Cartesian product of fuzzy sets	100
2.9.3	Extension principle	101
2.10	Composition and inference	105
2.10.1	Projection	105
2.10.2	Cylindrical extension	109
2.10.3	Join	115
2.10.4	Composition	116
2.10.4.1	Sup-product composition	116
2.10.5	Compositional rule of inference	117
2.10.5.1	Composition through matrix multiplication	119
2.10.6	Properties of composition	121
2.10.6.1	Sup-t composition	121
2.10.6.2	Inf-s composition	121
2.10.6.3	Commutativity	121
2.10.6.4	Associativity	122
2.10.6.5	Distributivity	123
2.10.6.6	DeMorgan's Laws	123
2.10.6.7	Inclusion	123
2.10.7	Extension principle	125
2.11	Considerations of fuzzy decision-making	126
2.11.1	Extensions to fuzzy decision-making	127
2.12	Summary	128
	Problems	128
	References	135
Chapter 3	Fuzzy logic control	137
3.1	Introduction	137
3.2	Background	138
3.3	Basics of fuzzy control	141
3.3.1	Steps of fuzzy logic control	145
3.3.2	Composition using individual rules	146

3.3.3	Defuzzification	151
3.3.3.1	Centroid method	151
3.3.3.2	Mean of maxima method	151
3.3.3.3	Threshold methods	152
3.3.3.4	Comparison of the defuzzification methods	152
3.3.4	Fuzzification	153
3.3.4.1	Singleton method	154
3.3.4.2	Triangular function method	154
3.3.4.3	Gaussian function method	155
3.3.4.4	Discrete case of fuzzification	156
3.3.5	Fuzzy control surface	156
3.3.6	Extensions of Mamdani fuzzy control	162
3.4	Fuzzy control architectures	162
3.4.1	Hierarchical fuzzy systems	164
3.4.2	Hierarchical model	166
3.4.2.1	Feedback/filter modules	168
3.4.2.2	Functional/control modules	169
3.4.3	Effect of information processing	169
3.4.4	Effect of signal combination on fuzziness	171
3.4.5	Decision table approach for a fuzzy tuner	172
3.5	Properties of fuzzy control	180
3.5.1	Fuzzy controller requirements	180
3.5.2	Completeness	181
3.5.3	Continuity	181
3.5.4	Consistency	182
3.5.5	Rule validity	185
3.5.6	Rule interaction	185
3.5.7	Rule base decoupling	186
3.5.7.1	Decision-making through a coupled rule base	187
3.5.7.2	Decision-making through an uncoupled rule base	189
3.5.7.3	Equivalence condition	190
3.6	Robustness and stability	191
3.6.1	Fuzzy dynamic systems	191
3.6.2	Stability of fuzzy systems	192
3.6.2.1	Traditional approach to stability analysis	192
3.6.2.2	Composition approach to stability analysis	195
3.6.3	Eigen-fuzzy sets	200
3.6.3.1	Iterative method	200
3.7	Summary	202
	Problems	202
	References	219

<b>Part II</b>	<b>Connectionist modeling and neural networks</b>	<b>221</b>
Chapter 4	Fundamentals of artificial neural networks	223
4.1	Introduction	223
4.2	Learning and acquisition of knowledge	224
4.2.1	Symbolic learning	224
4.2.2	Numerical learning	224
4.3	Features of artificial neural networks	226
4.3.1	Neural network topologies	227
4.3.1.1	The feedforward topology	227
4.3.1.2	The recurrent topology	227
4.3.2	Neural network activation functions	228
4.3.3	Neural network learning algorithms	230
4.3.3.1	Supervised learning	230
4.3.3.2	Unsupervised learning	231
4.3.3.3	Reinforcement learning	232
4.4	Fundamentals of connectionist modeling	233
4.4.1	McCulloch–Pitts models	233
4.4.2	Perceptron	234
4.4.3	Adaline	243
4.4.4	Madaline	244
4.5	Summary	246
	Problems	246
	References	248
Chapter 5	Major classes of neural networks	249
5.1	Introduction	249
5.2	The multilayer perceptron	250
5.2.1	Topology	250
5.2.2	Backpropagation learning algorithm	250
5.2.3	Momentum	260
5.2.4	Applications and limitations of MLP	262
5.3	Radial basis function networks	263
5.3.1	Topology	263
5.3.2	Learning algorithm for RBF	264
5.3.3	Applications	267
5.4	Kohonen's self-organizing network	268
5.4.1	Topology	268
5.4.2	Learning algorithm	269
5.4.3	Applications	273
5.5	The Hopfield network	274
5.5.1	Topology	274
5.5.2	Learning algorithm	276
5.5.3	Applications of Hopfield networks	281

5.6	Industrial and commercial applications of ANN	281
5.6.1	Neural networks for process monitoring and optimal control	282
5.6.2	Neural networks in semiconductor manufacturing processes	282
5.6.3	Neural networks for power systems	284
5.6.4	Neural networks in robotics	285
5.6.5	Neural networks in communications	286
5.6.6	Neural networks in decision fusion and pattern recognition	288
5.7	Summary	289
	Problems	290
	References	293
<b>Chapter 6</b>	<b>Dynamic neural networks and their applications to control and chaos prediction</b>	<b>299</b>
6.1	Introduction	299
6.2	Background	300
6.2.1	Basic concepts of recurrent networks	300
6.2.2	The dynamics of recurrent neural networks	301
6.2.3	Architecture	301
6.3	Training algorithms	304
6.3.1	Backpropagation through time (BPTT)	304
6.3.2	Real-time backpropagation learning	305
6.4	Fields of applications of RNN	306
6.5	Dynamic neural networks for identification and control	307
6.5.1	Background	307
6.5.2	Conventional approaches for identification and control	308
6.5.2.1	Systems identification	310
6.5.2.2	Adaptive control	311
6.6	Neural network-based control approaches	313
6.6.1	Neural networks for identification	315
6.6.2	Neural networks for control	320
6.6.2.1	Supervised control	320
6.6.2.2	Inverse control	321
6.6.2.3	Neuro-adaptive control	322
6.7	Dynamic neural networks for chaos time series prediction	323
6.7.1	Background	324
6.7.2	Conventional techniques for chaos system prediction and control	324

6.7.3	Artificial neural networks for chaos prediction	325
6.7.3.1	Conventional feedforward networks	325
6.7.3.2	Recurrent neural networks (RNNs)-based predictors	327
6.8	Summary	330
	Problems	330
	References	332
 Chapter 7 Neuro-fuzzy systems		337
7.1	Introduction	337
7.2	Background	338
7.3	Architectures of neuro-fuzzy systems	339
7.3.1	Cooperative neuro-fuzzy systems	340
7.3.1.1	Neural networks for determining membership functions	341
7.3.1.2	Adeli–Hung algorithm (AHA)	342
7.3.1.3	Learning fuzzy rules using neural networks	343
7.3.1.4	Learning in fuzzy systems using neural networks	344
7.3.1.5	Identifying weighted fuzzy rules using neural networks	344
7.3.2	Neural network-driven fuzzy reasoning	344
7.3.3	Hybrid neuro-fuzzy systems	345
7.3.3.1	Architecture of hybrid neuro-fuzzy systems	346
7.3.3.2	Five-layer neuro-fuzzy systems	347
7.3.3.3	Four-layer neuro-fuzzy systems (ANFIS)	350
7.3.3.4	Three-layer neuro-fuzzy approximator	350
7.4	Construction of neuro-fuzzy systems	355
7.4.1	Structure identification phase	355
7.4.1.1	Grid-type partitioning	355
7.4.1.2	Clustering	356
7.4.1.3	Scatter partitioning	357
7.4.2	Parameter learning phase	357
7.4.2.1	The backpropagation learning algorithm	358
7.4.2.2	Hybrid learning algorithms	359
7.5	Summary	359
	Problems	359
	References	361

<b>Part III</b>	<b>Evolutionary and soft computing</b>	<b>363</b>
Chapter 8	Evolutionary computing	365
8.1	Introduction	365
8.2	Overview of evolutionary computing	366
8.2.1	Evolutionary programming	369
8.2.2	Evolutionary strategies	370
8.2.3	Genetic programming	370
8.2.4	Genetic algorithms	371
8.3	Genetic algorithms and optimization	372
8.3.1	Genotype	373
8.3.2	Fitness function	374
8.4	The schema theorem: the fundamental theorem of genetic algorithms	375
8.5	Genetic algorithm operators	376
8.5.1	Selection	377
8.5.2	Crossover	377
8.5.3	Mutation	378
8.5.4	Mode of operation of GAs	378
8.5.5	Steps for implementing GAs	381
8.5.6	Search process in GAs	381
8.6	Integration of genetic algorithms with neural networks	388
8.6.1	Use of GAs for ANN input selection	388
8.6.2	Using GA for NN learning	389
8.7	Integration of genetic algorithms with fuzzy logic	390
8.8	Known issues in GAs	391
8.8.1	Local minima and premature convergence	391
8.8.2	Mutation interference	392
8.8.3	Deception	392
8.8.4	Epistasis	392
8.9	Population-based incremental learning	393
8.9.1	Basics of PBIL	393
8.9.2	Generating the population	393
8.9.3	PBIL algorithm	394
8.9.4	PBIL and learning rate	395
8.10	Evolutionary strategies	395
8.11	ES applications	400
8.11.1	Parameter estimation	400
8.11.2	Image processing and computer vision systems	400
8.11.3	Task scheduling by ES	400
8.11.4	Mobile manipulator path planning by ES	400
8.11.5	Car automation using ES	401
8.12	Summary	401
	Problems	401
	References	402

<b>Part IV</b>	<b>Applications and case studies</b>	<b>405</b>
Chapter 9	Soft computing for smart machine design	407
9.1	Introduction	407
9.1.1	Intelligent machines	408
9.1.2	Intelligent control	408
9.1.3	Hierarchical architecture	409
9.1.4	Development steps	411
9.2	Controller tuning	413
9.2.1	Problem formulation	414
9.2.1.1	Rule base	415
9.2.1.2	Compositional rule of inference	417
9.2.2	Tuning procedure	418
9.2.2.1	Rule dissociation	418
9.2.2.2	Resolution relations	419
9.2.2.3	Tuning inference	421
9.2.2.4	Accuracy versus fuzzy resolution	421
9.2.3	Illustrative example	422
9.2.3.1	Resolution relation	423
9.2.3.2	Stability region	426
9.2.3.3	Tuning results	427
9.3	Supervisory control of a fish processing machine	427
9.3.1	Machine features	430
9.3.2	Supervisory control system	432
9.3.3	Information preprocessing	435
9.3.3.1	Image preprocessing	435
9.3.3.2	Servomotor response preprocessing	436
9.3.3.3	Cutter load preprocessing	439
9.3.3.4	Conveyor speed preprocessing	443
9.3.4	Knowledge-based decision-making	443
9.3.4.1	Knowledge acquisition	444
9.3.4.2	Decision-making	446
9.3.4.3	Servo tuning	449
9.3.4.4	Product quality assessment	452
9.3.4.5	Machine tuning	453
9.3.5	System implementation	453
9.3.5.1	System modules	455
9.3.5.2	User interface of the machine	456
9.3.6	Performance testing	457
9.3.6.1	Servomotor tuning examples	457
9.3.6.2	Machine tuning example	461
9.3.6.3	Product quality assessment example	463
9.4	Summary	467
	Problems	467
	References	472

xvi	Chapter 10 Tools of soft computing in real-world applications	475
	Case study 1: Expert parameter tuning of DC motor controller	476
	Case study 2: Stabilizing control of a high-order power system by neural adaptive feedback linearization	497
	Case study 3: Soft computing tools for solving a class of facilities layout planning problem	510
	Case study 4: Mobile position estimation using an RBF network in CDMA cellular systems	522
	Case study 5: Learning-based resource optimization in ATM networks	533
	<i>Index</i>	555

# Preface

*Soft computing differs from conventional (hard) computing in that, unlike hard computing, it is tolerant of imprecision, uncertainty, and partial truth. In effect, the role model for soft computing is the human mind. The guiding principle of soft computing is: exploit the tolerance for imprecision, uncertainty, and partial truth to achieve tractability, robustness, and low solution cost.*

*Lotfi Zadeh  
UC Berkeley*

## Focus of the textbook

Soft computing concerns the use of theories of fuzzy logic, neural networks, and evolutionary computing to solve real-world problems that cannot be satisfactorily solved using conventional crisp computing techniques. Representation and processing of human knowledge, qualitative and approximate reasoning, computational intelligence, computing with words, and biological models of problem solving and optimization form key characteristics of soft computing, and are directly related to intelligent systems and applications. In recent years there has been rapid growth in the development and implementation of soft computing techniques in a wide range of applications, particularly those related to science and engineering. This book draws upon this vast body of existing knowledge, including the contributions of the present authors, and presents a comprehensive and cohesive treatment of the subject of soft computing from both analytical and practical points of view. It is particularly suitable as a textbook for senior undergraduate and graduate-level courses in such subjects as fuzzy logic, neural networks, evolutionary computing, intelligent machines, and intelligent control, in view of treatment and presentation. While sufficient theory of each subject is presented, it is offered in a simplified manner for the benefit of the students. Furthermore, a vast array of illustrative examples, end-of-chapter problems, exercises, projects,

and worked case studies representing a wide range of applications in the fields of science and engineering is presented to complement the theory and the techniques. Advanced topics and future challenges are addressed as well, with the researchers in the field in mind. The introductory material, application-oriented techniques, and case studies should be particularly useful to practicing professionals.

## Contents of the book

Generally speaking, soft computing represents a set of computational intelligence-based methodologies which are used to deal effectively with systems characterized by complex structures and possibly subjected to incomplete knowledge and ill-defined dynamics. These methodologies and tools can assist in the design, development, and operation of intelligent systems/machines which possess the behaviors of adaptation, learning, and autonomous operation. They involve computational techniques based on fuzzy set theory, connectionist modeling and evolutionary computing. Depending on the type of application they are used for, these tools could be implemented in either integrated or stand-alone schemes. Soft computing techniques allow the system designer to take advantage of the knowledge accumulated by the system either in a linguistic form or in data form, to continually learn from the operating experience, and to make use of state-of-the-art evolutionary computing algorithms to optimize the operation.

The book has five main parts. The first part presents tools of soft computing and the design of intelligent machines along with an outline of the fundamentals of fuzzy set theory, fuzzy logic, and its applications including intelligent control. Part II concentrates on connectionist modeling and its various aspects. Part III discusses general concepts of evolutionary computing and its applications. Part IV uses the tools of soft computing to design an intelligent machine. Part V tackles and solves a number of worked case studies using the tools developed throughout the textbook.

Part I is composed of three chapters. Chapter 1 introduces the notion of machine intelligence and soft computing tools and their merits in designing intelligent systems. Chapter 2 presents the fundamentals of fuzzy logic systems. Chapter 3 discusses major aspects of fuzzy inference mechanisms. It also provides the main ingredients for designing fuzzy logic controllers and several of their variants and applications.

Part II comprises four chapters, which describe neural networks and their integration into dynamic systems and fuzzy modeling. In particular, Chapter 4 introduces artificial neural networks (ANN) as a fundamental tool of soft computing with learning capabilities. Chapter 5 describes in some detail four major classes of ANN and provides an exhaustive review of applications of ANN in several sectors of industry. Chapter 6 introduces the notion of dynamic neural networks and outlines their use in dynamic processes and chaotic time series prediction. Chapter 7 presents a unified framework for

tackling in an integrated manner fuzzy logic systems and neural network structures.

Part III is composed of Chapter 8, which primarily introduces the notion of evolutionary computing and its application in science and engineering. In fact, this chapter represents the main foundation for another tool of soft computing known as evolutionary computing or genetic algorithms. It also includes means for integrating evolutionary computing approaches with those of fuzzy logic systems and neural networks.

Part IV of the book consists of the two remaining chapters, Chapter 9 and Chapter 10. In Chapter 9, we use the tools of soft computing that are developed throughout the textbook to design an intelligent machine. The specific example used here is a complex industrial process pertaining to an advanced automation cell for fish cutting. Chapter 10 complements the aspect of intelligent design and discusses the implementation of the associated tools in a number of case studies involving real-world applications in various fields of science and engineering.

## Possible course structures

The material in the book is suitable for a number of courses at the undergraduate and graduate levels. Some possibilities are given below.

- Fuzzy logic and applications (senior undergraduate or introductory graduate-level course): Chapters 1, 2, 3, and 9.
- Neural networks and applications (senior undergraduate or graduate-level course): Chapters 1, 4, 5, 6, 7, and 10.
- Soft computing and applications (senior undergraduate or graduate-level course): Chapters 1, 2, 4, 5, 7, and 8.
- Intelligent control (senior undergraduate or graduate-level course): Chapters 1, 2, 3, 4, 5, 6, 9, and 10.

A Solutions Manual giving detailed solutions to many of the end-of-chapter problems is available from [www.booksites.net/karray](http://www.booksites.net/karray), for the benefit of the instructors who adopt the textbook. Moreover, several Matlab files pertaining to some figures within the text can be found on the website, along with other supporting material, related websites of interest and updates to the book.



# Acknowledgements

The authors wish to express their appreciation to Mr Keith Mansfield, Acquisitions Editor, Pearson Education, for his great enthusiasm and support for the project. They wish to acknowledge as well the dedication and commitment of Ms Karen McLaren, Senior Editor, Mr Owen Knight, Editorial Assistant, and Ms Annette Abel. The Pearson Education team has been truly helpful at all levels and they deserve our sincere thanks.

The first author would like to thank warmly all those who have encouraged and supported him during the write-up of the manuscript. In particular, the author greatly appreciates the very insightful discussions and feedback he got on several occasions from Dr Lotfi Zadeh, the man fondly known around the world as the father of fuzzy logic. The author would like to thank as well many of his colleagues from around the world for their support and helpful feedback during the writing of the manuscript. To name a few: Dr Mohamed Kamel, Dr Otman Basir, Dr Madan Gupta, Dr Fathi Ghorbel, Dr Rogelio Soto, Dr Mo Jamshidi, and Dr Elmer Dadios. Very warm thanks are also due to some of the author's past and current students who gave feedback on the material on several occasions and who have assisted him directly or indirectly during the write-up: Dr Salah AlSharhan, Dr Wail Gueaieb, Dr Kingsley Fregene, Dr Nayer Wanas, Dr Federico Gudea, Dr Ravi Ravishandran, Dr Khariyah Youssef, Mr James Wang, Mr Insop Song, Mr Kevin Lee, Mr Wael Bazzi. The author is also very much indebted to his family for the strong support they have provided while putting together the material of the manuscript.

The second author wishes to acknowledge the vision and guidance provided by Dr Lotfi Zadeh (the father of fuzzy logic), Dr Mo Jamshidi (a renowned authority in soft computing and the author's friend), and Dr Madan Gupta (a renowned authority in soft computing and the author's friend). He wishes to express his gratitude to Sir Alistair MacFarlane (formerly Professor and Head of Information Engineering at the University of Cambridge) for providing him the opportunity to carry out research at Cambridge, particularly in the application of fuzzy control in robotics, when the field was not quite mature. Many of the applications incorporated in the

work were carried out in collaboration with the author's research associates and graduate students, too numerous to acknowledge by name here. The second author is grateful as well to his family for bearing much of the burden of the home while he was busy preparing this manuscript.

# List of acronyms

ADALINE	Adaptive linear element
AI	Artificial intelligence
AND	AND logic function
ANFIS	Adaptive-network-based fuzzy inference system
ANN (NN)	Artificial neural network (neural network)
ARMA(X)	Autoregressive moving average (with exogenous inputs)
ART	Adaptive resonance theory
BCS	Backward chaining system
BPL	Backpropagation learning
BPTT	Backpropagation through time
CDMA	Code division multiple access
CRI	Conjunction rule of inference
EP	Evolutionary programming
ES	Evolutionary strategies
FFNN	Feedforward neural networks
FIR	Finite duration impulse response
FIS	Fuzzy inferencing system
FLC	Fuzzy logic controller
FLT	Feedback linearization technique
FPS	Forward production system
G(MP)	Generalized ( <i>modus ponens</i> )
G(MT)	Generalized ( <i>modus tollens</i> )
GA	Genetic algorithms
GP	Genetic programming
HN	Hopefield network
HSRI	Hypothetical syllogism rule of inference
K(SOM)	Kohonen (self-organizing map)
KBS	Knowledge-based system
LMS(E)	Least mean square (error)
LQG	Linear quadratic Gaussian
LVQ	Learning vector quantization
MADALINE	Many adaline
MNN	Modular neural networks

---

MPRI	<i>Modus ponens</i> rule of inference
MRAC	Model-referenced adaptive control
MTRI	<i>Modus tollens</i> rule of inference
NARMA(X)	Nonlinear autoregressive moving average (with exogenous inputs)
NFS	Neuro-fuzzy systems
NOR	NOT-OR logic function
OR	OR logic function
PID	Proportional integral derivative (control)
RBFN	Radical basis function network
RMS(E)	Root mean square (error)
RNN	Recurrent neural network
SSE	Sum squared error
T-norm	Triangular norm
T-conorm	Triangular conorm
TDNN	Time-delay neural networks
XNOR	Exclusive NOR logic function
XOR	Exclusive OR logic function

# Fuzzy logic and fuzzy control



# Introduction to intelligent systems and soft computing

## Chapter outline

1.1	Introduction	3
1.2	Intelligent systems	3
1.3	Knowledge-based systems	12
1.4	Knowledge representation and processing	26
1.5	Soft computing	38
1.6	Summary	51
	Problems	51
	References	54

## 1.1 Introduction

This chapter provides a general introduction to the subjects of machine intelligence, knowledge-based systems, and soft computing, and forms a common foundation for the remaining chapters of the book. The concept of artificial intelligence is critically discussed. The linkage between intelligence and approximation is examined. Several concepts of approximation are presented and an analytical basis for them is given. Knowledge-based intelligent systems are discussed in some detail. Their architectures and methods of representing and processing knowledge are examined. Expert systems, which represent a practical class of knowledge-based systems, are studied. Techniques of soft computing, particularly fuzzy logic, neural networks, genetic algorithms, and probabilistic reasoning, are outlined. The presentation given here is intended to be neither exhaustive nor highly mathematical. A more rigorous treatment of some of the topics introduced here is found in other chapters.

## 1.2 Intelligent systems

Intelligent systems generally have a capacity to acquire and apply knowledge in an “intelligent” manner and have the capabilities of *perception, reasoning,*

*learning*, and making inferences (or decisions) from *incomplete information*. From historical times, humans have been successful in developing systems, which have supported our existence, advancement, and the quality of life. A variety of engineering, scientific, medical, legal, business, agricultural, and educational systems have been designed, developed, and implemented for our benefit. A feature that is indispensable in these systems is the generation of outputs, based on some inputs and the nature of the system itself. The inputs to a system may include information as well as tangible items, and the outputs may include decisions as well as physical products.

### Example 1.1

A typical input variable is identified for each of the following examples of dynamic systems:

- (a) Human body: neuroelectric pulses
- (b) Company: information
- (c) Power plant: fuel rate
- (d) Automobile: steering wheel movement
- (e) Robot: voltage to joint motor.

Possible output variables for each of these systems are:

- (a) Muscle contraction, body movements
- (b) Decisions, finished products
- (c) Electric power, pollution rate
- (d) Front wheel turn, direction of heading
- (e) Joint motions, effector motion.

Even when the outputs of a system are not decisions themselves, quite commonly, decision-making is involved in generating those outputs. The process of decision-making may range from numerical computations (e.g., in generating control actions in a low-level control algorithm) to complex, knowledge-based decision-making using natural (human) and artificial (machine) intelligence, as in modern decision support systems. This book primarily concerns the latter. Engineering systems such as household appliances and consumer electronics, transportation systems, manufacturing operations, factories, chemical plants, power generating systems, food processing facilities, and heating, ventilating, and air conditioning (HVAC) systems are increasingly becoming complex ill-defined. Explicit modeling of these would be a considerably difficult task. In some situations, even experimental modeling (or model identification) is not feasible because the required inputs and outputs of the system are not accessible or measurable. Even when modeling is possible, the models could be so complex that accurate algorithms for

decision-making (e.g., generation of control actions) based on these models could considerably increase the costs of computing and hardware, and seriously reduce the operating speed, thereby slowing the process and introducing unacceptable time lags and possibly instabilities. Knowledge-based systems with “intelligent” decision-making capabilities are tremendously beneficial in these systems.

Future generations of industrial machinery, plants, and decision support systems may be expected to carry out round-the-clock operation, with minimal human intervention, in manufacturing products or providing services. It will be necessary that these systems maintain consistency and repeatability of operation and cope with disturbances and unexpected variations within the system, its operating environment, and performance objectives. In essence, these systems should have the capability to accommodate rapid reconfiguration and adaptation. For example, a production machine should be able to quickly cope with variations ranging from design changes for an existing product to the introduction of an entirely new product line. This will call for tremendous flexibility and some level of autonomous operation in automated machines, which translates into a need for a higher degree of intelligence in the supporting devices. Smart systems will exhibit an increased presence and significance in a wide variety of applications. Products with a “brain” are found, for example, in household appliances, consumer electronics, transportation systems, industrial processes, manufacturing systems, and services. There is clearly a need to incorporate a greater degree of intelligence and a higher level of autonomy into automated machines. This will require the appropriate integration of such devices as sensors, actuators, and controllers, which themselves may have to be “intelligent” and, furthermore, appropriately distributed throughout the system. Design, development, production, and operation of intelligent machines have been made possible today through ongoing research and development in the field of intelligent systems and control.

### 1.2.1 Machine intelligence

In the context of machine intelligence, the term *machine* is normally used to denote a computer or a computational machine. In this sense *machine intelligence* and *computational intelligence* are synonymous. Historically *machine intelligence*, *computer intelligence* and *artificial intelligence* (AI) also have come to mean the same thing. Artificial intelligence can be defined as the science of making machines do things that would require intelligence if done by humans. Conventional AI has relied heavily on symbolic manipulation for the processing of descriptive information and “knowledge” in realizing a degree of intelligent behavior. The knowledge itself may be represented in a special high-level language. The decisions that are made through processing of such “artificial” knowledge, perhaps in response to data such as sensory signals, should possess characteristics of intelligent decisions of a human. Knowledge-based systems and related expert systems are an outcome of the efforts made by the AI community in their pursuit of intelligent computers and intelligent machines. The field of soft computing is important here,

developments of which have taken a somewhat different path from traditional AI, yet they have contributed to the general goal of realizing intelligent machines, thereby broadening the meaning of machine intelligence. This topic, which forms the foundation of the present book, will be treated in detail in subsequent chapters.

The first digital computer may be traced back to the development of universal Turing machines in the mid-1930s, but it was in the 1960s that significant activity in the field of artificial intelligence began with the objective of developing computers that could think like humans. Just like neurons in the brain, the hardware and software of a computer are themselves not intelligent, yet it has been demonstrated that a computer may be programmed to demonstrate some intelligent characteristics of a human. According to Marvin Minsky of the Massachusetts Institute of Technology, “artificial intelligence is the science of making machines do things that would require intelligence if done by men.” In this context it is the outward characteristics (outputs) of a computer that may be termed “intelligent” rather than any similarity of the computer programs to how a human processes information. In particular, it is not the physical makeup that is considered in ascertaining whether the machine is intelligent. Specifically, an analogy may be made here with intelligent behavior of a human whose brain, made up of neurons, is a physical entity that assists in realizing that behavior. A considerable effort has gone into the development of machines that somewhat mimic humans in their actions. Because it is the thought process that leads to intelligent actions, substantial effort in AI has been directed at the development of artificial means to mimic the human thought process. This effort is somewhat related to cognitive science. The field of study that deals with analyzing and modeling of the information processing capabilities of a human is known as *cognitive science* and is important in AI. Nevertheless, in developing an intelligent machine it is not essential to master cognitive science.

### 1.2.2 Meaning of intelligence

It is useful to first explore the meaning of the term itself. A complete yet simple definition of *intelligence* is not available. At the basic or atomic level of implementation, what a computer does is quite procedural and cannot be considered intelligent. Similarly, one may argue that the 20 billion neurons in a human brain perform operations that are hardly intelligent when taken individually. It is these same neurons, however, that govern the intelligent behavior of a human. In particular, the attributes of knowledge acquisition – making logical inferences, learning, and dealing with incomplete or qualitative information and uncertainty – are all associated with human intelligence. Since intelligence is a soft and complex concept, it is rather unrealistic to expect a precise definition for the term. Narrow definitions can result in misleading interpretations, similar to how a group of blind people defined a proverbial elephant.

A definition for *intelligence*, then, has to be closely related to the characteristics or outward “appearance” of an intelligent behavior. This is called

description through characterization. There exist many characteristics that can be termed *intelligent*, and partly for this reason it has not been possible to give a precise definition for intelligence. It is the outward characteristics of a system that qualify it to be classified as being intelligent – for example, possession of a memory, ability to learn and thereby gain knowledge and expertise, ability to satisfactorily deal with unexpected and unfamiliar situations, and ability to reason, deduce, and infer from incomplete information. In particular, pattern recognition and classification play an important role in intelligent processing of information. For example, an intelligent system may be able to recognize and acquire useful information from an object that is aged or distorted, having been previously familiar with the original, undistorted object. Somewhat related are the concepts of approximation, which may lead one to treat the ability to approximate as also a characteristic of intelligence. It is commonly accepted that an intelligent system possesses one or more of the following characteristics and capabilities:

- Sensory perception
- Pattern recognition
- Learning and knowledge acquisition
- Inference from incomplete information
- Inference from qualitative or approximate information
- Ability to deal with unfamiliar situations
- Adaptability to new yet related situations (through expectational knowledge)
- Inductive reasoning
- Common sense
- Display of emotions
- Inventiveness.

Even though significant advances have been made, particularly in the first five capabilities listed above, the current generation of intelligent machines does not claim to have all these capabilities, particularly the last three in the list. Clearly this list of items does not represent a formal definition of an intelligent system. It is known, however, that humans possess these characteristics and capabilities and that humans are intelligent beings. A handwriting recognition system is a practical example of an intelligent system. The underlying problem cannot be solved through simple template matching, which does not require intelligence. Because handwriting can vary temporally for the same person, due to various practical shortcomings such as missing characters, errors, nonrepeatability, sensory restrictions, and noise, a handwriting recognition system has to deal with incomplete information and unfamiliar characters and should possess capabilities of learning, pattern recognition, and approximate reasoning, which will assist in carrying out intelligent functions of the system.

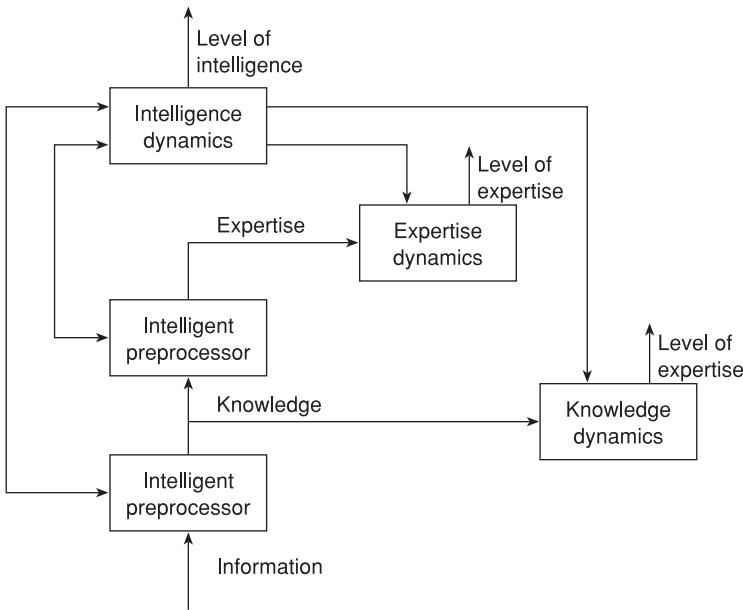


Figure 1.1: A schematic model for the dynamics of intelligence

### 1.2.3 Dynamics of intelligence

Every day we come across *information*, for example in the form of newspapers, radio and TV broadcasts, sensory data, books, charts, and computer files. Information itself does not represent knowledge. We have to “acquire” knowledge, say through experience and by learning. In this sense, information is subjected to a form of high-level processing to gain knowledge and hence we may interpret knowledge as *structured information*. Specialized and enhanced knowledge may be termed *expertise*. Here again knowledge is subjected to high-level processing to acquire expertise.

A simplified and qualitative model for the dynamics of intelligence is shown in Figure 1.1. This model illustrates the hierarchical nature of the associated processes as well. In the model of Figure 1.1, intelligent preprocessing is identified as the operation that converts information into knowledge and knowledge into expertise. A degree of intelligence is needed to carry out these preprocessing operations, and simultaneously the intelligence itself will be broadened or enhanced through preprocessing, as indicated by the bi-directional activity lines. Knowledge may be interpreted as “structured information” that is acquired by an intelligent object, and expertise as “specialized knowledge.” In this sense, then, preprocessing operations will carry out structuring and acquisition of information in gaining knowledge and will use specialized knowledge in gaining expertise. Note that knowledge and expertise can depreciate, and be outdated or lost, in the absence of continuous usage, learning, and updating (i.e., intelligent preprocessing). The blocks

of “Knowledge dynamics” and “Expertise dynamics” account for such appreciation and depreciation, in the present simplified model. Similarly, the block of “Intelligence dynamics” allows for variations in the level of intelligence.

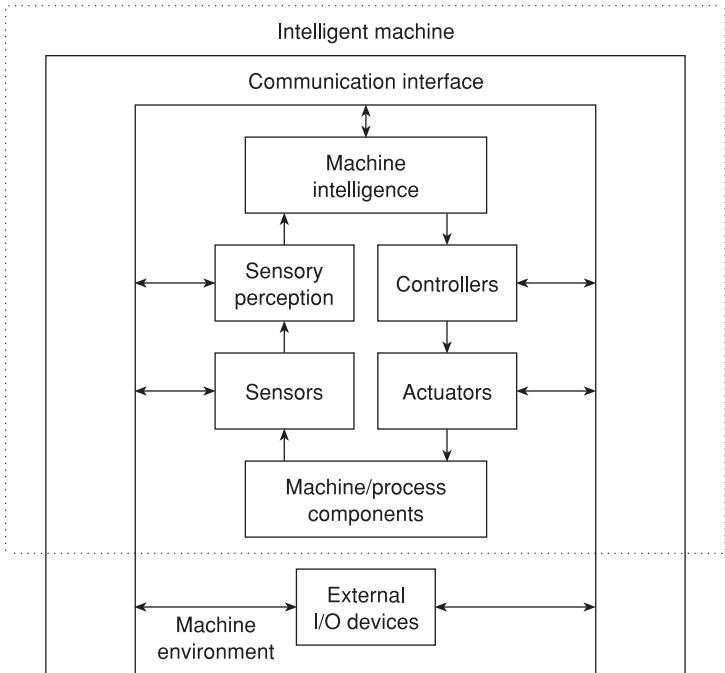
Knowledge and expertise form the “knowledge base” of an intelligent system. Then, with any new information that may arrive from external sources such as sensors and human interfaces, the knowledge-based system can make perceptions (sensory perception) and new inferences or decisions by interpreting the meaning and implications of the new information within the capabilities of the existing knowledge base. The associated decision-making task is an intelligent processing activity, which in turn may lead to enhancement, refinement, and updating of the knowledge base.

#### 1.2.4 Intelligent machines

An intelligent machine is a machine that can exhibit one or more intelligent characteristics of a human. As much as neurons themselves in a brain are not intelligent but certain behaviors that are affected by those neurons are, the physical elements of a machine are not intelligent but the machine can be programmed to behave in an intelligent manner. An intelligent machine embodies machine intelligence. An *intelligent machine*, however, may take a broader meaning than an *intelligent computer*.

The term may be used to represent any process, plant, system, device, or machinery that possesses machine intelligence. Sensors, actuators, and controllers will be integral components of such a process and will work co-operatively in making the behavior of the process intelligent. Sensing with understanding or “feeling” what is sensed is known as *sensory perception*, and this is very important for intelligent behavior. Humans use vision, smell, hearing, and touch (tactile sensing) in the context of their intelligent behavior. Intelligent machines too should possess some degree of sensory perception. The “mind” of an intelligent machine is represented by machine intelligence. For proper functioning of an intelligent machine it should have effective communication links between various components. An intelligent machine may consist of a structural system for carrying out the intended functions of the machine. Computers that can be programmed to perform “intelligent” tasks such as playing chess or understanding a natural language are known to employ AI techniques for these purposes, and may be classified as intelligent machines either by themselves or in conjunction with other structural devices such as robotic hands and visual, sonic, chemical, and tactile interfaces. By taking these various requirements into consideration, a general-purpose structure of an intelligent machine is given in Figure 1.2.

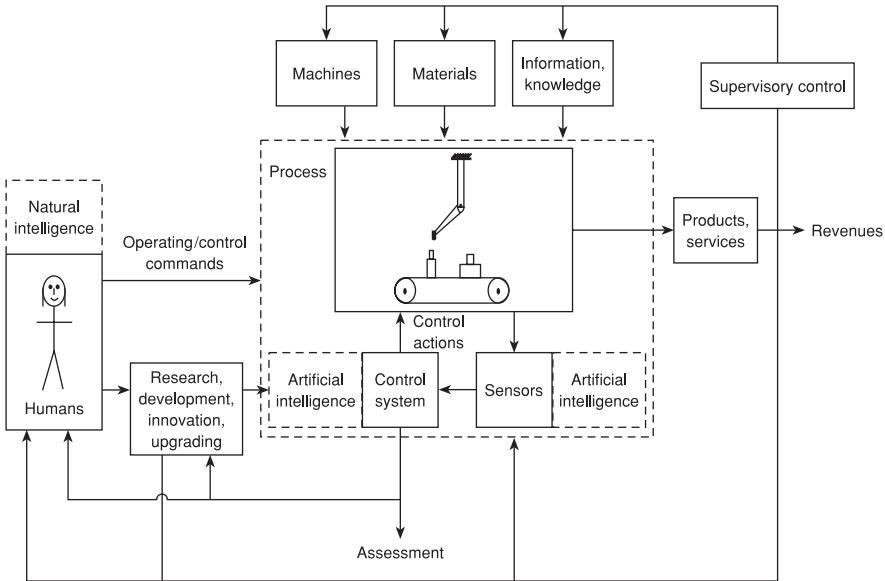
In broad terms, an intelligent machine may be viewed to consist of a *knowledge system* and a *structural system*. The knowledge system effects and manages intelligent behavior of the machine, loosely analogous to the brain, and consists of various knowledge sources and reasoning strategies. The structural system consists of physical hardware and devices that are necessary to perform the machine objectives yet do not necessarily need a knowledge



**Figure 1.2:** An intelligent machine

system for their individual functions. Sensors, actuators, controllers (nonintelligent), communication interfaces, mechanical devices, and other physical components fall into this category. The broad division of the structure of an intelligent machine, as mentioned above, is primarily functional rather than physical. In particular the knowledge system may be distributed throughout the machine, and individual components themselves may be interpreted as being “intelligent” as well (for example, intelligent sensors, intelligent controllers, intelligent multi-agent systems). It needs to be emphasized that an actual implementation of an intelligent system will be domain specific, and much more detail than what is alluded to in Figure 1.2 may have to be incorporated into the system structure. Even from the viewpoint of system efficiency, domain-specific and special-purpose implementations are preferred over general-purpose systems.

The structure of an intelligent process is given in Figure 1.3, where the important components of the system and the flow of signals/information are indicated. Sensing, sensory perception, actuation, control, and knowledge-based decision-making are crucial for the proper performance of an intelligent process. In the general structure shown in Figure 1.3, machines, materials, information, and knowledge are used to generate products and services according to some specifications. Revenue generation will be an



**Figure 1.3:** The structure of an intelligent process

economic goal of the process. Quality of the products and services, speed of operation, reliability, environmental considerations, and so on may be included as well within the overall goals. Assessment of the products and services and also the performance of the process would be crucial here. Since the process models and goals may not be mathematically precise, and since there could be unexpected and unknown inputs and disturbances to the process, concepts of artificial intelligence and soft computing would be useful in the tasks of modeling, control, assessment, adaptation, and modification of the system.

No matter what the goals of an intelligent machine are, sensing will play a significant role in its operation. For example, if a machine possesses human emotions, one could convincingly classify it as an intelligent machine. But the mechanism of decision-making and control that would generate such characteristics needs to have a means of sensing these characteristics based on the machine operation. Both intelligent sensing and intelligent control are important in this respect. In particular, an intelligent sensor may consist of a conventional sensor with further processing of signals and knowledge-based “intelligent” decision-making, as indicated in Figure 1.3.

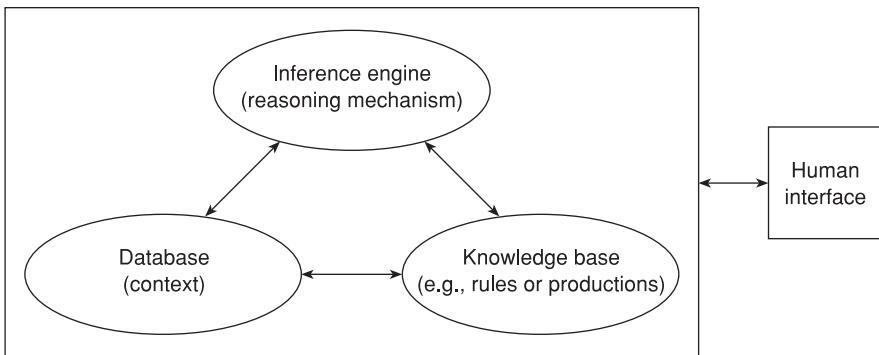
Even the most sophisticated intelligent machines could benefit from a human interface, which would provide an interface for “natural” intelligence. The interaction with human experts can be made both off line and on line. System upgrading and innovation may result from research and development, and can be implemented through such communication interfaces.

### 1.3 Knowledge-based systems

The method of knowledge-based systems is perhaps the most notable AI paradigm to have found widespread application, particularly in process and manufacturing industries, and medical, business, education, and legal fields. This technique can represent a controller or a decision support system according to a number of architectures, and is able to handle heuristics and empirical knowledge. A basic definition of a knowledge-based system is a comprehensive computer program which solves problems within a limited and specific field, using data on the problem, knowledge related to the problem, and “intelligent” decision-making capabilities. It typically emulates the problem-solving behavior of a human expert in the field; specifically, it tries to represent and execute the knowledge and reasoning strategy of one or more experts. In this context, it is known as an expert system. This is different from the conventional algorithmic programming technique where the solution to a problem can be obtained by executing a prescribed set of steps according to a flowchart. Algorithmic programming offers fixed solutions to fixed (or “hard” or “crisp”) problems. On the other hand, a knowledge-based system is more suited to accomplish tasks where the nature of the problems and solutions is not well defined or not known beforehand. In industrial plants (say, in process control), for example, there are situations involving a significant number of variable factors such as changing plant characteristics, unexpected disturbances, system wear and tear, and different fault and alarm combinations, where the approach of knowledge-based systems has certain advantages and flexibility which can make the method particularly appropriate or attractive in such systems. For example, in the process control industry, systems with a large number of operating rules and constraints requiring complex logic are commonplace. As a result, knowledge-based systems have great potential for successful application in the process control industry for handling a variety of applications requiring inference-making (or decision-making), including process monitoring, fault diagnosis, alarm management, and process scheduling and optimization. In some developments, knowledge-based systems have also been incorporated as part of the primary feedback loop, offering enhanced automatic and optimized control.

Figure 1.4 depicts the basic structure of a knowledge-based system (KBS). There are four basic components: the knowledge base (or knowledge source), database, inference engine, and the system interface. A knowledge-based system is able to make perceptions (e.g., sensory perception) and new inferences or decisions using its reasoning mechanism (inference engine) by interpreting the meaning and implications of the new information within the capabilities of the existing knowledge base. These inferences may form the outputs of the knowledge-based systems. The associated decision-making task is an intelligent processing activity, which in turn may lead to enhancement, refinement, and updating of the knowledge base itself.

Central to a knowledge-based system is the so-called knowledge base. The knowledge base contains knowledge and expertise in the specific



**Figure 1.4:** The structure of a knowledge-based system

domain, particularly domain-specific facts and heuristics useful for solving problems in the domain. The knowledge source is implemented as an identifiable and separate part of the program. This may be represented in various forms, and commonly as a set of if-then rules (called productions).

The second basic component is the database, which is primarily a short-term memory. The database contains the current status of the problem, inference states, and the history of solutions to date. New information that is generated or arrives from external sources such as sensors and human interfaces is stored in the database. This data represents the “context” of the decision-making process.

The inference engine is the “driver” program of a knowledge-based system. Depending on the data in the database, the inference engine applies and operates on the knowledge in the knowledge source to solve problems and arrive at conclusions. Specifically, it traverses the knowledge base, in response to observations and other inputs provided to it from the external world, and possibly previous inferences and results from the KBS itself, and will identify one or more possible outcomes or conclusions. This task of making inferences for arriving at solutions will involve “processing” of knowledge. It follows that representation and processing of knowledge are central to the functioning of a KBS. The inference engine will incorporate various inference mechanisms, for example forward chaining, backward chaining, and various search algorithms. The data structure selected for the specific form of knowledge representation determines the nature of the program created as an inference engine.

The fourth basic component of a KBS is the system interface (or user interface), which is provided for the external world (or user) to interact with the overall system, to browse through the knowledge source, to edit the knowledge (e.g., rules) and for many other interactive tasks. Keyboards, screen displays, sensors, transducers, and even output from other computer programs, including expert systems, usually provide the interface between a knowledge-based system and the external world.

Well-developed knowledge-based systems will also have additional capabilities such as a knowledge acquisition facility, and an explanation feature which can explain its reasoning and the decisions to the user. In the special case of production systems (i.e., rule based systems), the knowledge base consists of a set of rules written, for example, as an ASCII file. Therefore the knowledge acquisition facility that is required for these systems is often merely an editor. In systems based on other forms of representation, the knowledge acquisition facility will generally be an integral part of the KBS and can be used only with that system.

In the problem of knowledge-based decision-making, sensory information and any other available data on the process are evaluated against a knowledge base concerning the specific application, making use of an inference-making procedure. Typically, this procedure consists of some form of “matching” of the abstracted data with the knowledge base. In particular, for a knowledge base  $K$  and a set of data or information  $D$  on the particular process, the procedure of arriving at a decision or inference  $I$  may be expressed as

$$I = M[P(D), K] \quad (1.1)$$

in which the “preprocessing operator”  $P(.)$  converts the context information on the process into a form that is compatible with  $K$ . Knowledge-base matching of the preprocessed information is performed using the matching operation  $M[.]$ . Knowledge-based systems such as rule based systems and fuzzy decision-making systems in particular follow this model.

### 1.3.1 Architectures of knowledge-based systems

A knowledge-based system may be organized and developed according to one or a combination of several architectures. Three commonly used architectures are:

- (1) Production systems
- (2) Frame-based systems
- (3) Blackboard systems.

These three architectures have some commonalities and also some distinguishing features. Let us outline some basic features of these architectures.

### 1.3.2 Production systems

Production systems are rule based systems, which are appropriate for the representation and processing of knowledge (i.e., knowledge-based decision-making) in problem solutions associated with artificial intelligence. An expert system, which will be discussed in detail in a subsequent section, is a good

example of a production system. In a production system, knowledge is represented by a set of rules (or *productions*) stored in the knowledge base. The database contains the current data (or context) of the process. The inference engine is the reasoning mechanism, which controls rule matching, coordinates and organizes the sequence of steps used in solving a particular problem, and resolves any conflicts.

The operation (processing) of a typical rule based system proceeds as follows: new data are generated (say, from sensors or external commands) and stored in appropriate locations within the database of the system. This is the new context. The inference engine tries to match the new data with the condition part (i.e., the *if* part or the *antecedent*) of the rules in the knowledge base. This is called rule searching. If the condition part of a rule matches the data, that rule is “fired,” which generates an action dictated by the action part (i.e., the *then* part or the *consequent*) of the rule. In fact, firing of a rule amounts to the generation (inference) of new facts, and this in turn may form a context that will lead to the satisfaction (firing) of other rules.

Knowledge representation using a set of if-then rules is not an unfamiliar concept. For example, a maintenance or troubleshooting manual of a machine (e.g., automobile) contains such rules, perhaps in tabular form. Also, a production system may be used as a simple model for human reasoning: sensory data fire rules in the short-term memory, which will lead to the firing of more complex rules in the long-term memory.

### Example 1.2

Consider a knowledge base for selecting a control technique, as given by the following set of rules:

*If the plant is linear and uncoupled then use Control Category A.*  
*If the plant is linear and coupled then use Control Category B.*  
*If the plant is nonlinear use Control Category C.*  
*If Category A and a plant model is known then use Subgroup 1.*  
*If Category B and a plant model is known then use Subgroup 2.*  
*If Subgroup 1 and high model uncertainty then use H-infinity control.*

...  
etc.

Now suppose that the database received the following context:

*Linear*  
*Coupled*  
*Model Available*  
*Model Uncertainty High*

In this case, the first two items in the context will fire Rule 1. The results will form a new item in the context. This new item, along with the third item in the old context, will fire Rule 5. The result, along with the last item of the old context, will fire Rule 6, which will lead to the selection of  $H \propto$  control, in this situation.

### 1.3.2.1 Reasoning strategies

Two strategies are available to perform reasoning and make inferences in a production system:

- (1) Forward chaining
- (2) Backward chaining.

*Forward chaining* is a data-driven search method. Here, the rule base is searched to match an *if* part of a rule (condition) with known facts or data (context), and if a match is detected, that rule is fired (i.e., the *then* part or “action” part of the rule is activated). Obviously, this is a direct strategy and is a bottom-up approach. Actions could include creation, deletion, and updating of data in the database. As seen in the previous example on control-technique selection, one action can lead to firing of one or more new rules. The inference engine is responsible for sequencing the matching (searching) and action cycles. A production system that uses forward chaining is termed a *forward production system* (FPS). This type of system is particularly useful in knowledge-based control. In the previous example, forward chaining was tacitly used.

In *backward chaining*, a top-down search process, a hypothesized conclusion is matched with rules in the knowledge base in order to determine the context (facts) that supports the particular conclusion. If enough facts support a hypothesis, the hypothesis is accepted. Backward chaining is particularly useful in situations that call for diagnosis and theorem proving, and generally in applications where a logical explanation has to be attached to each action (say, in an expert system). A production system that uses backward chaining is called a *backward chaining system* (BCS). In the previous example on control-technique selection, if one had hypothesized the selection of another control technique (say, the LQG method) and proceeded backwards in searching the rule base for the context information that satisfies this hypothesis, one would arrive at a set of requirements (premises). But these premises would not match the entire context of the situation (specifically, the context “model uncertainty high” will not be matched) and hence the hypothesis would be false. Then, one should try another hypothesis and proceed similarly, until a suitable match is obtained.

### 1.3.2.2 Conflict resolution methods

When the context data are matched with the condition parts of the rules in a rule base, it may be possible that more than one rule is satisfied. The set of rules that is satisfied in this manner is termed the *conflict set*. A method of conflict resolution has to be invoked to select from the conflict set the rule that would be fired. Methods of conflict resolution include the following:

- (1) First match
- (2) Toughest match
- (3) Privileged match
- (4) Most recent match.

In the first method, the very first rule that is satisfied during searching will be fired. This is a very simple strategy but may not produce the best performance in general. In the second method, the rule with the most condition elements, within the conflict set, will be fired. For example, suppose that the conflict set has the following two rules:

*If the temperature is high then increase the coolant flow rate.*

*If the temperature is high and the coolant flow rate is maximum, then shut down the plant.*

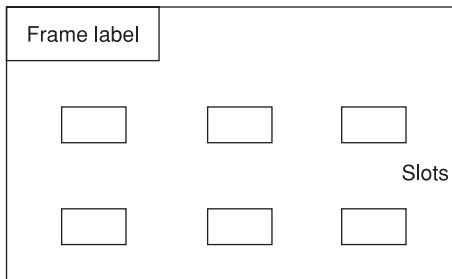
Here, the toughest match is the second rule.

The rules in a rule base may be assigned various weightings and priorities depending on their significance. The privileged match is the rule with the highest priority, within the conflict set. For example, a priority may be assigned in proportion to the toughness of the match. In this case the methods 2 and 3 listed above are identical. Alternatively, a priority may be assigned to a rule on the basis of the significance or consequences of its acts.

The most recent match is the rule in the conflict set whose condition part satisfies the most recent entries of data. In this method, a higher priority is given to more recently arrived data in the database.

### 1.3.3 Frame-based systems

A frame is another arrangement for representing and processing knowledge in artificial intelligence (AI) applications. Consequently, it can form the basis of the architecture of a knowledge-based system. A frame is a data structure developed to represent *expectational knowledge*, i.e., knowledge of what to expect when entering a given situation for the first time. Common sense knowledge or general knowledge can be represented in this way and new information (new frames) can be interpreted using old information (old frames) in a hierarchical manner. A frame may contain *context knowledge* (facts) and *action knowledge* (cause–effect relations). The frames that contain action rules (i.e., *procedural knowledge*) are called *action frames*. The other



**Figure 1.5:** Schematic representation of a frame

types of frames that define the object distribution (i.e., *factual knowledge*) are called *situational frames*.

A frame consists of a *frame label* and a set of *slots*, as schematically shown in Figure 1.5. Each slot stores either a set of statements (rules) or another frame that is at the next lower level in the hierarchy than the present level. In this manner, knowledge may be represented at several hierarchical levels in different degrees of detail (i.e., *resolution*). Generation of a knowledge base using frames is an evolutionary procedure. First, a generic frame (i.e., a *template* or a *schematic frame*) is activated within the knowledge-base computer. This schematic frame will have a default label and a standard number of default sets. Subsequent steps are as follows:

- (1) Give a name (label) to the frame.
- (2) Name the slots to represent objects contained within the frame.
- (3) Open the slots one at a time, and either program the set of action rules or define the lower-level frames as in Step 2.
- (4) Repeat Step 3 as many times as required, depending on the required number of levels in the hierarchy.

### Example 1.3

Consider a frame representing a manufacturing workcell, as shown in Figure 1.6. Note that the workcell consists of a four-axis robot, milling machine, conveyor, positioning table, inspection station, and a cell-host computer. They are represented by slots in the cell frame. Each of these slots contains a situational frame. For example, the robot slot stores the robot frame as shown. The cell host also has a data slot, which stores the data files that are needed for the operation of the cell. The programs slot contains an action frame, which carries various procedures needed in the operation of the cell. For example, the programs that are needed for carrying out various cell tasks, through proper coordination of the cell components, monitoring, and control, are carried by the "Host programs" frame.

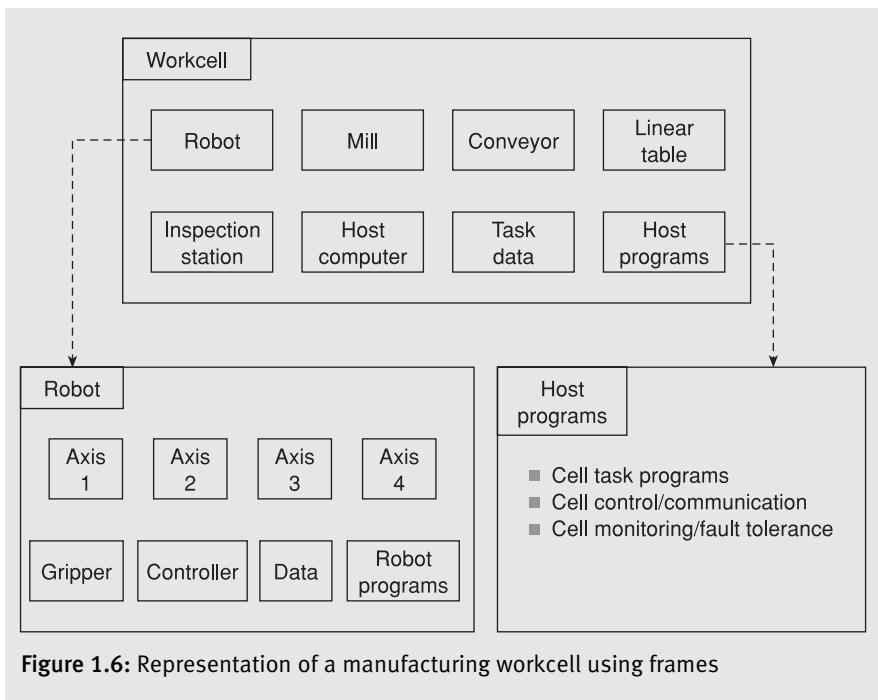


Figure 1.6: Representation of a manufacturing workcell using frames

So far we have considered knowledge representation using frames. Now let us address reasoning or knowledge processing, which is essential for problem solution through frames. Reasoning using frames is done as follows. An external input (say, operator command or a sensor signal) or a program action triggers a frame according to some heuristics. Then, slots in the frame are matched with *context* (i.e., current data including sensory inputs), which may lead to a possible updating of slot values in the current frame, and to triggering of subsequent frames, once the conditions are matched. Finally the appropriate procedures are executed by the action frames.

To illustrate this procedure, consider once again the manufacturing workcell problem shown in Figure 1.6. During operation of the reasoning mechanism the knowledge-base computer opens a workcell frame. There will be both manual and automatic data inputs to the computer indicating the cell components (e.g., robot, mill). The system matches the actual components with the slots of the frame. If there is no match and if the system has another workcell frame, that frame is matched. In this manner, the closest match is chosen and updated to obtain an exact match for all frames at all levels. Then the action slots of the frame at the highest level (cell level) are activated. This will initiate the cell actions, which in turn trigger the cell-component actions. Note that in a hierarchical system of this type, the knowledge and actions needed at a higher level are generally more “intelligent” than what are needed at a lower level. But the resolution (i.e., degree of detail) of the information needed at a higher level is lower than at a lower level. For example, the

coordination of a multi-component workcell is an activity that would require far more intelligence than the servo control of an axis of a robot. But the information for the former is needed at a considerably lower bandwidth and resolution than that for the latter.

### 1.3.4 Blackboard systems

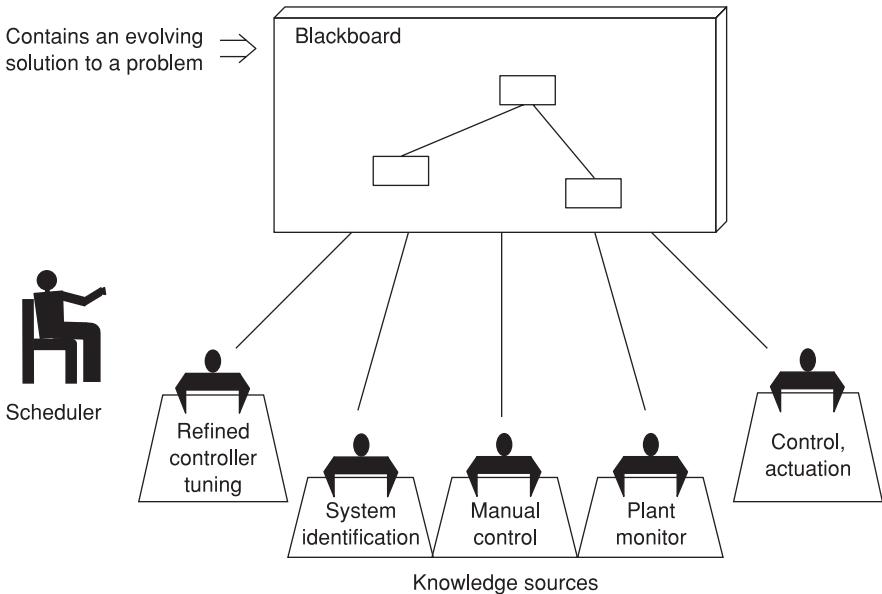
A suitable structure for a knowledge-based system would be a blackboard architecture, which is a cooperative problem-solving architecture. It consists of a global database called a *blackboard*, several intelligent modules called *knowledge sources*, and a *control unit*, which manages the operation of the system. The main feature of the blackboard architecture is the common data region (blackboard), which is shared by and visible to the entire system. In particular, the database is shared by the knowledge sources.

A blackboard-based system has the flexibility of accommodating different types of knowledge sources, with associated methods of knowledge representation and processing. Generally the knowledge sources are not arranged in a hierarchical manner and will cooperate as equal partners (specialists) in making a knowledge-based decision. The knowledge sources interact with the shared data region under the supervision of the control unit.

When the data in the blackboard change, which corresponds to a change in the context (data condition), the knowledge sources would be triggered in an opportunistic manner and an appropriate decision would be made. That decision could then result in further changes to the blackboard data and subsequent triggering of other knowledge sources. Data may be changed by external means (for example, through the user interface) as well as by knowledge-source actions. External data entering the system go directly to the blackboard. Also, the user interface is linked to the blackboard.

The operation of a blackboard-based system is controlled by its control unit. Specifically, when an updating of data occurs in the blackboard, the control unit will trigger the appropriate knowledge source, which will result in the execution of some reasoning procedure and, possibly, generation of new information. This architecture is generally fast, and particularly suitable for real-time control applications. A blackboard may consist of more than one layer (hierarchical level), each consisting of a subsystem, itself having a blackboard-system architecture. In this manner, a hierarchical structure can be organized for the system. Hybrid systems consisting of subsystems having production and frame-based architectures can be developed as well.

Figure 1.7 shows a blackboard application of controller tuning for a process plant. A knowledge-based controller is implemented in a hierarchical architecture. The hardware consists of a computer workstation connected through an IEEE-488 parallel bus to a front-end dedicated controller board, which is built around the Intel 8096 microcontroller. This board constitutes the lower layer of the hierarchy where actual real-time control is performed. The microcontroller computes and sends the control signal to the plant at every sampling interval. At the upper level of the hierarchy is the workstation where the knowledge-based system is implemented. The workstation runs on



**Figure 1.7:** A blackboard architecture

UNIX and has three processes running concurrently. The knowledge-based process oversees and supervises the microcontroller on line. This involves starting and stopping of particular algorithms, calculating algorithm parameters, analyzing results from identification algorithms, bumpless transfer from one algorithm to another, and reacting correctly to alarms from the monitoring algorithm. The user-interface process provides man-machine interaction for the operator, while the IEEE-488 communication process exchanges information with the microcontroller board.

The knowledge-based system is implemented using a commercial expert system shell (NEXPERT Object), which is a hybrid shell that has rule-based and frame-based knowledge representation. The knowledge-based process is implemented in a blackboard architecture and has six knowledge sources as depicted in Figure 1.7. The manual control knowledge source supervises the manual control operation and is also responsible for gathering open-loop process information like open-loop gain and process noise during manual control. The system identification knowledge source supervises the on-off (relay) control operation, gathers process information such as input-output data, ultimate gain and ultimate period, and carries out experimental modelling. The refined controller tuning knowledge source supervises the fine-tuning of the automatic controller. The plant monitor knowledge source examines plant response information. It contains heuristics to determine the achievable performance of the plant and the controller and decides whether the plant performance is acceptable. The controller and actuation knowledge source oversees the control and actuation tasks for the plant. The scheduler knowledge source takes overall charge of the blackboard. It contains rules

that keep track of changes on the blackboard and decides which knowledge source to activate.

### 1.3.5 Object-oriented programming

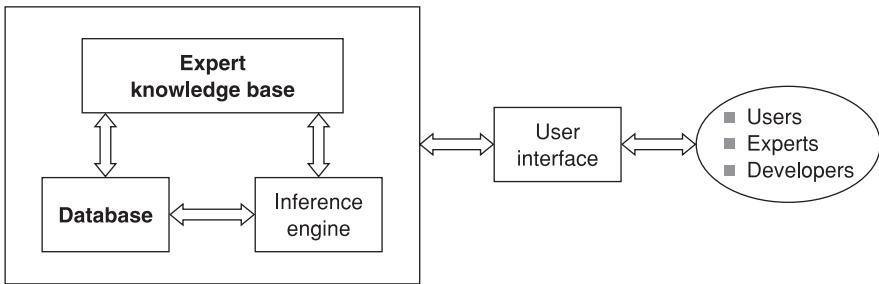
We have noticed that the architecture of a knowledge-based system is characterized by the presence of multiple objects that are interlinked. Specifically, one or more distinct knowledge sources together with separate memory regions or databases would be present. In this manner, the domain knowledge is isolated from the data. Hence, the data structure of one object may be modified without affecting the knowledge base and the remaining objects of the system. This is different from the traditional programs written in procedural languages such as FORTRAN, PASCAL, and C. In a conventional program, written in a procedural language, the instructions and data structures are integrated together throughout the entire program. Hence even a small change to a data structure could make the program nonfunctional, clearly indicating an advantage of the object-oriented approach.

Many early knowledge-based systems were implemented in a symbolic language like Lisp and Prolog. However, with proper design, there can be a clear separation between the knowledge source and the inference engine, and this has led to the development of many so-called knowledge-based system shells or frameworks. A knowledge-based system shell is just an empty knowledge-based system without any domain knowledge. It provides an inference engine and a knowledge representation structure that can be used as a programming tool for developing knowledge-based systems in different application areas.

The architectures of knowledge-based systems provide a modularity that results from the explicit representation of knowledge as a distinct component of the system, in the form of a knowledge source. The knowledge source can be built incrementally and is relatively easy to expand. Each module or object may be developed separately, and appropriately interconnected, using *object-oriented programming*, to generate and implement a very complex logic system, in a convenient manner. A frame-based architecture with multiple “slots” is particularly amenable to object-oriented programming. In contrast, in conventional algorithmic programming, a complex logic system will result in program codes with huge numbers of nested if-then-else loops. This causes the program to become difficult to understand and not amenable to maintenance, debugging, or modular development. As a result, object-oriented extensions such as C++ and CLOS to conventional programming languages such as C and LISP have been developed.

### 1.3.6 Expert systems

Expert systems are a class of production systems, which are typically used in a *consultation mode*, and provide an alternative to consulting a human expert. They deal with complex knowledge for which real expertise is required, and this knowledge is typically acquired from human experts.



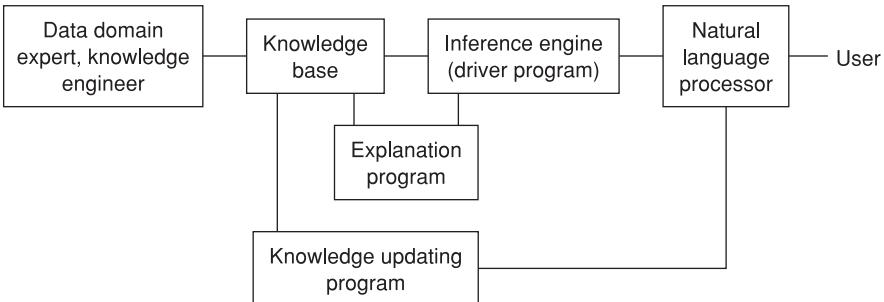
**Figure 1.8:** Components of an expert system

Consequently, it is unrealistic to expect an expert system for an application where human expertise would not be available. Knowledge-based systems are a general form of expert systems in view of the fact that they are quite general in application and not limited to mimicking the role of a human expert. Furthermore, very often they acquire knowledge from non-human information sources.

An expert system is defined as a software system with high symbolic and descriptive information content, which can simulate the performance of a human expert in a specific field or domain. Since an expert system is a special type of knowledge-based system, it contains the traditional constituents of the latter such as a knowledge base, a database, an inference engine, and a human/machine interface, as indicated in Figure 1.8.

The system interface is used for both development, particularly knowledge acquisition, and utilization of the expert system. The knowledge base of an expert system embodies human knowledge and understanding, somewhat imitating a human expert in the particular domain of expertise (e.g., specialist, engineer, scientist, doctor, lawyer, financial consultant). The inference engine is a “driver” program that traverses the knowledge base in response to observations and other inputs from the external world, and possibly previous inferences and results from the expert system itself, and will identify one or more possible outcomes or conclusions. This task of making inferences for arriving at solutions will involve “processing” of knowledge. The data structure selected for the specific form of knowledge representation determines the nature of the program that serves as an inference engine. Monitors and keyboards, sensors and transducers, and even output from other computer programs including expert systems, usually provide communication links between an expert system and the external world.

An expert system is intended to take the place of a human expert in an advisory capacity. It can be used by a nonexpert to improve problem-solving abilities or by an expert to obtain supporting or corroborating information in her decision-making process. An expert system is typically used in a consultation mode. It receives problem descriptions (for example, in a question-answer form) and provides advice through knowledge-based reasoning. In addition to a dedicated *knowledge acquisition facility*, an expert system should be an *explanation facility*, because it should be able to explain its



**Figure 1.9:** Operational schematics of an expert system

inferences (advice). Operational schematics of an expert system are shown in Figure 1.9.

#### 1.3.6.1 Development of an expert system

The development of an expert system requires the combined effort of domain experts, who are human experts in the field of interest, and knowledge engineers, who acquire and represent/program the knowledge in a suitable form. This process will require a knowledge acquisition facility. Other engineers, programmers, etc. are needed for the development of the system interface, integration, and so on. The experts will be involved in defining the problem domain to be solved and in providing the necessary expertise and knowledge for the knowledge base that will lead to the solution of a class of problems in that domain. The knowledge engineers are involved in acquiring and representing the available knowledge and in the overall process of programming and debugging the system. They organize the acquired knowledge into the form required by the particular expert system tool. Forms of knowledge representation may include logic, production systems (rules), semantic scripts, semantic primitives, frames, and symbolic representations. Experts and knowledge engineers both will be involved in testing and evaluating the system.

Typically, what is commercially available for developing expert systems is an expert system “shell.” It will consist of the required software programs, but will not contain a knowledge base. It is the responsibility of the user, then, to organize the creation of the required knowledge base, which should satisfy the requirements of the system with respect to the form of knowledge representation that is used and the structure of the knowledge base.

In developing the knowledge base of an expert system, the experts involved may rely on published information, experimental findings, and any other available information in addition to their own knowledge. Testing and evaluation may involve trials where the experts pose problems to the expert system and evaluate the inferences made by the system. Many of these test problems may have solutions that are already known. Others may not be so clear cut and hence the responses of the expert system should be carefully scrutinized by the experts during the testing and evaluation phase. The tests should involve the ultimate users of the system as well. In this regard, the

features of the user interface, including simplicity, use of a common language, graphics, and voice facilities, are important.

#### 1.3.6.2 *Knowledge engineering*

A “knowledge engineer” gathers the expertise about a particular domain from one or more experts, and organizes that knowledge into the form required by the particular expert system tool that is to be used. Forms of knowledge representation may include logic, production system (rules), semantic scripts, semantic primitives, and frames and scripts. A typical expert system uses a data structure as the basis of the particular representation technique it implements. Consequently, the knowledge engineer needs to know the general form in which the knowledge is to be represented and the particular form of that representation required by the expert system itself. The knowledge that is “engineered” in this manner forms the knowledge base of the expert system.

In summary, the tasks of knowledge engineering include the following:

- (1) Acquisition of knowledge that is pertinent, from different sources (experts, literature, media, etc.)
- (2) Interpretation and integration of the knowledge (from various sources and in different forms)
- (3) Representation of the knowledge within the knowledge-based system (suitable structure, language, etc. have to be chosen. Here, one should consider aspects of incomplete knowledge, presence of analytical models, accessibility of system variables for measurement, and availability of past experience)
- (4) Processing of knowledge for making inferences (this operation has to be compatible with the knowledge base, objectives of the system, etc. Speed of decision-making is crucial, particularly in real-time applications. The form of the inferences should be consistent with the system objectives)
- (5) System needs and constraints (accuracy, implications of incomplete knowledge, and cost of an incorrect inference)
- (6) Economic considerations (development cost; cost to the user in comparison to the benefits).

#### 1.3.6.3 *Applications*

An expert system may be equally useful to both an expert and a layperson. For example, it is difficult for one expert to possess complete knowledge in all aspects of a problem and, furthermore, the solutions to a problem can be quite complex. Then, the expert may turn to a good expert system that will provide the necessary solutions, which the expert could evaluate further (perhaps with the assistance of other experts) before adopting. Ideally, an expert system should have an “evolving” knowledge base, which has the capability to learn and continuously update itself.

Expert systems have been developed for a variety of applications such as education and training, medical diagnosis and prescription of treatment,

mineral exploration, interpretation of satellite imagery, financial advising, legal consultation, tax return preparation, system troubleshooting and maintenance, planning and scheduling, weather forecasting, troubleshooting, maintenance and operation of plants and machinery, and system control. An expert system that is used to supervise the control system of a plant is called a *control expert system*. Such expert systems are directly applicable at a high level of operation, for monitoring, supervision, diagnosis, and control of intelligent machines.

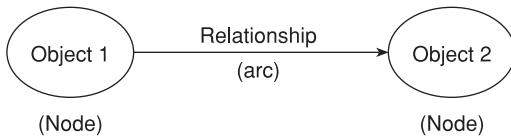
While the quality and performance of an expert system is highly dependent on the knowledge base that is contained within it, this knowledge alone does not constitute a powerful expert system. A poorly chosen formalism may limit the extent and nature of the knowledge; accordingly, the structure of knowledge representation plays an important role. The formalism that is employed should closely resemble how an expert uses knowledge and performs reasoning. The skill of both the domain expert and the knowledge engineer will be crucial here. The techniques of representation and processing will be dependent on the type of knowledge. The reasoning scheme that is employed will directly affect the search speed and how conflicts are resolved. In particular, retrieval and processing of information will not be fast or efficient without a good formalism for representing knowledge and a suitable inference-making scheme to process it. In real-time expert systems, speed of decision-making is a crucial factor, and it depends on hardware/software characteristics, the inference-making scheme, and the particular technique of knowledge representation.

Performance goals of the next-generation expert systems are:

- Automatic generation of code and knowledge representation
- Automatic learning and system enhancement from experience
- Voice recognition
- Communication through a natural language
- Automated translation of documents into knowledge bases
- Cooperative problem-solving architectures
- Generic problem-solving shells
- Multilevel reasoning systems.

## 1.4 Knowledge representation and processing

An appropriate representation of knowledge, including intuition and heuristic knowledge, is central to the development of *machine intelligence* and of knowledge-based systems. In a knowledge-based system two types of knowledge are needed: knowledge of the problem (problem representation or modeling) and knowledge regarding methods for solving the problem. Ways of representing and processing knowledge include: logic, semantic networks, frames, production systems, and fuzzy logic. Several of these approaches have been discussed before. The rest are examined next.



**Figure 1.10:** A segment of a semantic network

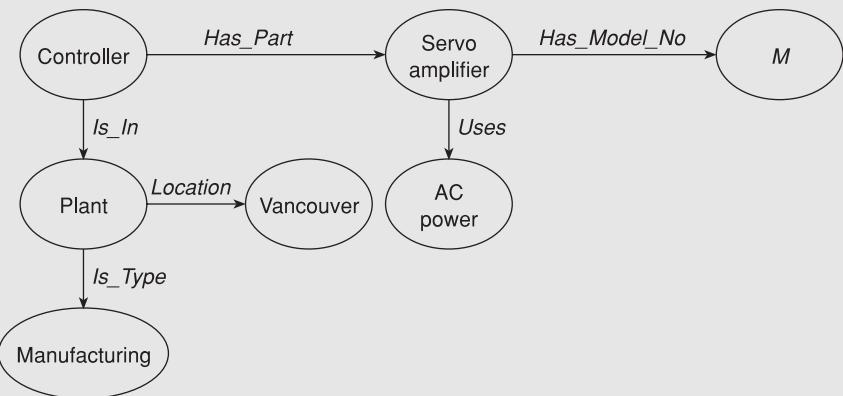
### 1.4.1 Semantic networks

Semantic networks are useful in the graphical representation and processing of knowledge. Here, knowledge objects are represented in a network, and their relationships are represented by *arcs* or lines linking the nodes. The arcs are “directed,” meaning they use arrows to indicate the direction of the relationship between the connecting objects. An arc represents a particular association between two objects. For example, a marriage can be represented by a “*married to*” arc joining a *husband object* and a *wife object*. Knowledge represented by a semantic network is processed using network searching procedures, usually starting with some available data and ending with a set of conclusions.

The nomenclature of a semantic network is described in Figure 1.10. Specifically, it shows the relationship of Object 1 to Object 2. The object at a node may be generic (e.g., “Plant”) or specific (e.g., “Plant 1”). Relationships represented by links can take many forms. Examples are: *IS\_A*, *HAS*, *CONTAINS*, and *HAS\_FEATURE*.

#### Example 1.4

To illustrate the concepts of semantic networks, consider the example shown in Figure 1.11.



**Figure 1.11:** An example of a semantic network

This semantic net represents the following knowledge:

- (1) The controller has a servo amp, which uses AC power.
- (2) The controller is in a plant, which is located in Vancouver.
- (3) It is a manufacturing plant.
- (4) The servo amplifier of the controller belongs to a specific model type.

Note that  $M$  is a generic model number. Based on some new information (say, a specified model number), the semantic network may be searched (processed) to generate new knowledge (e.g., to which plant the servo amp belongs). A semantic net takes a hierarchical structure when the “ $/s\_A$ ” relationship alone is used in its representation. This is just a special case of a general semantic net.

### 1.4.2 Crisp logic

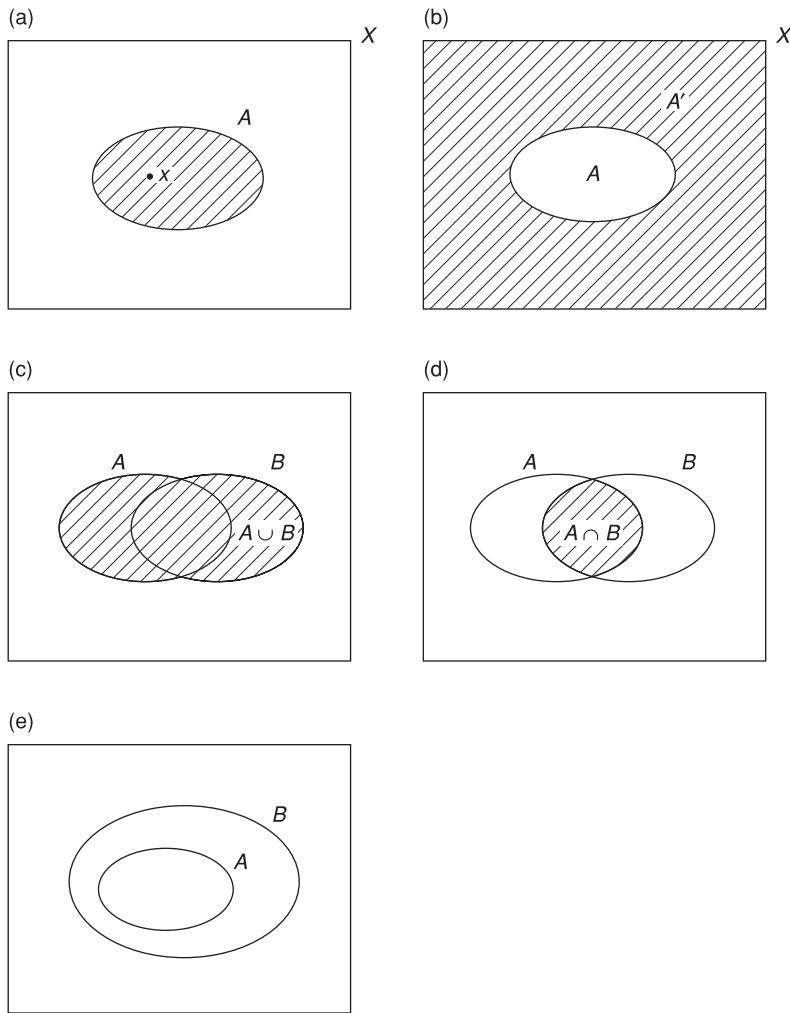
Logic is a useful technique of representing and processing knowledge, and is applicable in knowledge-based systems. In logic, knowledge is represented by statements called *propositions*, which may be joined together using *connectives*. The knowledge may be processed through reasoning, by the application of various laws of logic including an appropriate *rule of inference*. We shall limit our discussion to crisp, binary (i.e., *two-valued*) logic. Crisp sets and binary logic are analogous. Now we will present an introduction to crisp sets, and highlight the isomorphism that exists between crisp sets and binary (crisp) logic.

#### 1.4.2.1 Crisp sets

A crisp set is a collection of elements within a crisp boundary. Since there cannot be any elements on the boundary, this is called a “crisp” set. A set of this type may be graphically represented by a *Venn diagram*, as shown in Figure 1.12(a). Here  $A$  denotes a set. The *universal set* (or universe of discourse), as denoted by  $X$ , is the largest set that one could consider in a particular problem domain. It contains all possible elements, and there cannot be any elements outside it. The null set is the empty set, and is denoted by  $\emptyset$ . It does not have any elements. If an element  $x$  is inside set  $A$ , then  $x$  is called a member of  $A$ . This is represented by the notation  $x \in A$ . Of course, by definition,  $x \in X$ . If  $x$  is outside  $A$  (i.e.,  $x$  is not a member of  $A$ ), we write  $x \notin A$ . An example of a set is the group of people who live in Cincinnati who are over 50 years old. As another example,

$$A = \{a_1, a_2, \dots, a_n\} \quad (1.2)$$

represents the set containing the  $n$  elements  $a_1, a_2, \dots, a_n$ . As yet another example, the set of numbers greater than 50 may be denoted by



**Figure 1.12:** Some concepts of crisp sets: (a) membership, (b) complement, (c) union, (d) intersection, (e) subset (proper)

$$A = \{x | x > 50\} \quad (1.3)$$

Here, the expression that follows the symbol “|” gives the condition (or property) to which the elements of the set are subjected.

#### 1.4.2.2 Operations of sets

The basic operations involving sets are complement, union, and intersection.

**Complement:** A complement of a set  $A$  is the set formed by all the elements outside  $A$ . It is denoted by  $A'$  and is shown by the shaded area in Figure

1.12(b). In particular, the universal set  $X$  is the complement of the null set  $\emptyset$ , and vice versa.

**Union:** The union of two sets  $A$  and  $B$  is the set formed by all the elements in  $A$  and  $B$ . This is denoted by  $A \cup B$ , and is shown by the shaded area in Figure 1.12(c).

**Intersection:** The intersection of two sets  $A$  and  $B$  is the set formed by all the elements that are common to both  $A$  and  $B$ . This is denoted by  $A \cap B$ , and is shown by the shaded area in Figure 1.12(d).

**Subset:** A set  $A$  is a subset of another set  $B$  if all the elements in  $A$  are common to (i.e., contained in)  $B$ . This is denoted by  $A \subseteq B$ . This is equivalent to  $B \supseteq A$ . If  $A$  is a subset of  $B$  but  $B$  is not a subset of  $A$ , then  $A$  is said to be a “proper subset” of  $B$ , and is denoted by  $A \subset B$ . This is equivalent to  $B \supset A$ , and is shown by Figure 1.12(e).

The definitions given above assume that the sets involved belong to the same universe ( $X$ ).

#### 1.4.2.3 Logic

Conventional logic deals with statements called “propositions.” In binary (or *two-valued*) logic, a proposition can assume one of only two truth values: true( $T$ ), false( $F$ ). An example of a proposition would be “John is over 50 years old.” Now consider the following propositions:

- (1) Charcoal is white.
- (2) Snow is cold.
- (3) Temperature is above 60°C.

Here proposition 1 has the truth value  $F$ , and proposition 2 has the truth value  $T$ . But, for proposition 3 the truth value depends on the actual value of the temperature. Specifically, if the temperature is above 60°C the truth value is  $T$  and otherwise it is  $F$ .

In logic, knowledge is represented by propositions. A simple proposition does not usually make a knowledge base. Many propositions connected/modifies by logical connectives such as *AND*, *OR*, *NOT*, *EQUALS*, and *IMPLIES* may be needed. These basic logical operations are defined next. The *truth tables* of these connectives are shown in Table 1.1. Note that a truth table gives the truth values of a combined proposition in terms of the truth values of its individual components.

**Negation:** The negation of a proposition  $A$  is “NOT  $A$ ” and may be denoted as  $\bar{A}$  (also  $\sim A$ ). It is clear that when  $A$  is *TRUE*, then NOT  $A$  is *FALSE* and vice versa. These properties of negation may be expressed by a *truth table*, as shown in Table 1.1(a).

**Table 1.1:** Truth tables of some logical connectives(a) Negation (*NOT*)

$A$	$\bar{A}$
$T$	$F$
$F$	$T$

(b) Disjunction (*OR*)

$A$	$B$	$A \vee B$
$T$	$T$	$T$
$T$	$F$	$T$
$F$	$T$	$T$
$F$	$F$	$F$

(c) Conjunction (*AND*)

$A$	$B$	$A \wedge B$
$T$	$T$	$T$
$T$	$F$	$F$
$F$	$T$	$F$
$F$	$F$	$F$

(d) Implication (*IF-THEN*)

$A$	$B$	$A \rightarrow B$
$T$	$T$	$T$
$T$	$F$	$F$
$F$	$T$	$T$
$F$	$F$	$T$

As an example, consider the proposition “John is over 50 years old.” If John’s age is actually over 50 years, then this proposition is true. Otherwise it is false. Now consider the set of people who are more than 50 years old. If John is a member of this set, then the above proposition is true. If John is a member of the complement of the set, the above proposition is false. Then the negated proposition “John is not over 50 years old” becomes true. This shows that there is a correspondence between the “complement” operation in set theory and the “negation” operation in logic.

**Disjunction:** The disjunction of the two propositions  $A$  and  $B$  is “ $A$  OR  $B$ ” and is denoted by the symbol  $A \vee B$ . Its truth table is given in Table 1.1(b). In this case, the combined proposition is true if at least one of its constituents is true. Note that this is not the “Exclusive OR” where “ $A$  OR  $B$ ” is false also when both  $A$  and  $B$  are true, and is true only when either  $A$  is true or  $B$  is true. It is easy to see that the “disjunction” operation in logic corresponds to the “union” operation in sets.

**Conjunction:** The conjunction of two propositions  $A$  and  $B$  is “ $A$  AND  $B$ ” and is denoted by a symbol  $A \wedge B$ . Its truth table is given in Table 1.1(c). In this case the combined proposition is true if and only if both constituents are true. There is correspondence between “conjunction” in logic and “intersection” in set theory.

**Implication:** Consider two propositions  $A$  and  $B$ . The statement “ $A$  implies  $B$ ” is the same as “*IF A THEN B*.” This may be denoted by  $A \rightarrow B$ . Note that if both  $A$  and  $B$  are true then  $A \rightarrow B$  is true. If  $A$  is false, the statement “When  $A$  is true then  $B$  is also true” is not violated regardless of whether  $B$  is true or

false. But if  $A$  is true and  $B$  is false, then clearly the statement  $A \rightarrow B$  is false. These facts are represented by the truth table shown in Table 1.1(d).

### Example 1.5

Consider two propositions  $A$  and  $B$ . Form the truth table of the combined proposition (*NOT A*) OR  $B$ . Show that, in accordance with two-valued crisp logic, this proposition is identical to the statement “*IF A THEN B*.”

#### **Solution**

The truth of the combined proposition is given below.

$A$	$\bar{A}$	$B$	$\bar{A} \vee B$
$T$	$F$	$T$	$T$
$T$	$F$	$F$	$F$
$F$	$T$	$T$	$T$
$F$	$T$	$F$	$T$

Here we have used Tables 1.1(a) and (b). Note that the result is identical to Table 1.1(d). This equivalence is commonly exploited in logic associated with knowledge-based decision-making.

It is appropriate to mention here that the two propositions  $A$  and  $B$  are equivalent if  $A \rightarrow B$  and also  $B \rightarrow A$ . This may be denoted by either  $A \leftrightarrow B$  or  $A \equiv B$ . Note that the statement “ $A \leftrightarrow B$ ” is true either “if both  $A$  and  $B$  are true” or “if both  $A$  and  $B$  are false.”

In real life we commonly make use of many “shades” of truth. Binary logic may be extended to many-valued logic that can handle different *grades* of truth value in between  $T$  and  $F$ . In fact, fuzzy logic may be interpreted as a further generalization of this idea.

#### 1.4.2.4 Correspondence between sets and logic

As indicated before, there is an isomorphism between set theory and logic. For crisp sets and conventional two-valued logic, this correspondence is summarized in Table 1.2. Boolean algebra is the algebra of two-valued logic. The two values used are 1 (corresponding to *true*) and 0 (corresponding to *false*). Accordingly there is also a correspondence between the operations of Boolean algebra and logic, as indicated in Table 1.2. Note that the two values in Boolean algebra may represent any type of two states (e.g., *on* or *off* state; *presence* or *absence* state; *high* or *low* state; two voltage levels in *transistor-to-transistor logic*) in practical applications.

**Table 1.2: Isomorphism between set theory, logic, and Boolean algebra**

Set theory concept	Set theory notation	Binary logic concept	Binary logic notation	Boolean algebra notation
Universal set	$X$	(Always) true	(Always) $T$	1
Null set	$\emptyset$	(Always) false	(Always) $F$	0
Complement	$A'$	Negation (NOT)	$\bar{A}$ or $\sim A$	$\bar{A}$
Union	$A \cup B$	Disjunction (OR)	$A \vee B$	$A + B$
Intersection	$A \cap B$	Conjunction (AND)	$A \wedge B$	$A \cdot B$
Subset	$A \subseteq B$	Implication (if-then)	$A \rightarrow B$	$A \leq B$

#### 1.4.2.5 Logic processing (reasoning and inference)

Knowledge may be processed through *reasoning*, by the application of various laws of logic including an appropriate *rule of inference*, subjected to a given set of data (measurements, observations, external commands, previous decisions, etc.) to arrive at new inferences or decisions. In intelligent control, for example, a knowledge base is processed through reasoning, subjected to a given set of data (measurements, observations, external commands, previous decisions, etc.) to arrive at new control decisions.

Previously we have discussed the *representation* of knowledge in logic. Now let us address the *processing* of knowledge in logic. The typical end objective of knowledge processing here is to make inferences. This may involve the following steps:

- (1) Simplify the knowledge base by applying various laws of logic.
- (2) Substitute into the knowledge base any new information (including data and previous inferences).
- (3) Apply an appropriate rule of inference.

Note that these steps may be followed in any order and repeated any number of times, depending on the problem.

#### 1.4.2.6 Laws of logic

The laws of logic govern the truth-value equivalence of various types of logical expressions consisting of the connectives AND, OR, NOT, etc. They are valuable in simplifying (reducing) a logical knowledge base. Consider three propositions  $A$ ,  $B$ , and  $C$  whose truth values are not known. Also, suppose that  $X$  is a proposition that is always true ( $T$ ) and  $\phi$  is a proposition that is always false ( $F$ ). With this notation the important laws of logic are summarized in Table 1.3. It is either obvious or may be easily verified, perhaps using Table 1.1, that in each equation given in Table 1.3, the truth value of the left-hand side is equal to that of the right-hand side.

**Table 1.3: Some laws of logic**

Law	Truth value equivalence
Commutativity	$A \wedge B = B \wedge A$ $A \vee B = B \vee A$
Associativity	$(A \wedge B) \wedge C = A \wedge (B \wedge C)$ $(A \vee B) \vee C = A \vee (B \vee C)$
Distributivity	$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$ $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$
Absorption	$A \vee (A \wedge B) = A$ $A \wedge (A \vee B) = A$
Idempotency (Idem = same; potent = power) (Similar to unity or identity operation)	$A \vee A = A$ $A \wedge A = A$
Exclusion: Law of excluded middle Law of contradiction	$A \vee \bar{A} = X \equiv T$ $A \wedge \bar{A} = \phi \equiv F$
DeMorgan's Laws	$\overline{A \wedge B} = \bar{A} \vee \bar{B}$ $\overline{A \vee B} = \bar{A} \wedge \bar{B}$
Boundary conditions	$A \vee X = X \equiv T$ $A \wedge X = A$ $A \vee \phi = A$ $A \wedge \phi = \phi \equiv F$

**Example 1.6**

Consider the following two propositions:

$A$  = “Plant 1 is stable.”

$B$  = “Plant 2 is stable.”

Now consider the combined proposition

$\overline{A \wedge B}$  = “It is not true that both plants 1 and 2 are stable.”

and the combined proposition

$\bar{A} \vee \bar{B}$  = “Either Plant 1 is not stable or Plant 2 is not stable.”

Obviously, both these combined propositions have the same truth value (and the same linguistic meaning). This verifies the DeMorgan Law given in Table 1.3.

In conventional (crisp) logic,  $A \text{ OR } (\text{NOT } A) = T$ . That is, this combined proposition is always true, and hence the common “middle” is not excluded in forming the combined proposition. In some other types of logic (e.g., fuzzy logic, as we will see), this combined proposition is not always true, and the common “middle” of the two propositions will be excluded.

Similarly, in conventional (crisp) logic,  $A \text{ AND } (\text{NOT } A) = F$ . That is, this combined proposition is always false, and hence the two combined propositions are a contradiction. In some other types of logic (e.g., fuzzy logic, as we will see), this combined proposition is not a complete contradiction.

#### 1.4.2.7 Rules of inference

Rules of inference are used in reasoning associated with knowledge processing, to make inferences. Four types of rules of inference are given below:

- (1) Conjunction Rule of Inference (CRI)
- (2) Modus Ponens Rule of Inference (MPRI)
- (3) Modus Tollens Rule of Inference (MTRI)
- (4) Hypothetical Syllogism Rule of Inference (HSRI).

The meanings of these rules of inference are indicated in Table 1.4.

**Table 1.4: Some rules of inference in conventional logic**

Rule of inference	Operation
Conjunction	$(A, B) \Rightarrow A \wedge B$
Modus Ponens	$(A \wedge (A \rightarrow B)) \Rightarrow B$
Modus Tollens	$(\bar{B} \wedge (A \rightarrow B)) \Rightarrow \bar{A}$
Hypothetical Syllogism	$((A \rightarrow B) \wedge (B \rightarrow C)) \Rightarrow (A \rightarrow C)$

**Conjunction:** The CRI states that, if a proposition  $A$  is true and a second proposition  $B$  is true, then the combined proposition “ $A \text{ AND } B$ ” is also true. This seemingly simple and obvious fact has applications in rule-based reasoning.

#### Example 1.7

Consider a simple knowledge base of response characteristics consisting of just two propositions:

*The speed of response is high.  
The overshoot is excessive.*

Note that each statement is a proposition, which may or may not be true. Now suppose that we have some new data (say from a speed sensor and a position sensor), which indicate a high speed of response and also an excessive overshoot. With these data, each one of the two propositions becomes a *premise* (which is known to be true). Then by applying the CRI we can infer the following result:

*“The speed of response is high and the overshoot is excessive.”*

**Modus Ponens:** The MPRI states that if proposition  $A$  is true and also the implication  $A \rightarrow B$  holds, then it can be concluded that proposition  $B$  is also true. This rule of inference is much more widely applicable than the CRI in rule-based reasoning.

### Example 1.8

To illustrate the use of MPRI, suppose that a knowledge base has the rule:

*If the trajectory error exceeds 1.0 mm the robot performance is unacceptable.*

Also, suppose that during operation of the robot, a position sensor indicates that the trajectory error is 1.1 mm. Then by using this information and applying the MPRI the following inference can be made:

*“Robot performance is unacceptable.”*

Note that this inference may serve as data (new information) for another rule such as:

*“If the robot performance is unacceptable tune the controller”*

which may be processed by another application of MPRI.

**Modus Tollens:** The MTRI states that if proposition  $B$  is not true and also the implication  $A \rightarrow B$  holds, then it can be concluded that proposition  $A$  is also not true.

**Hypothetical Syllogism:** The term syllogism means “deductive reasoning” or “calculation with words.” The HSRI states that if the implication  $A \rightarrow B$  holds and also the implication  $B \rightarrow C$  holds, then it can be concluded that the implication  $A \rightarrow C$  also holds.

#### 1.4.2.8 Propositional calculus and predicate calculus

Propositional calculus is the branch of logic where propositions (i.e., statements that are either true or false) are used in logic “calculations.” Note that the term “calculus” in the present terminology denotes “the approach of calculation” and has nothing to do with differential or integral calculus. In the previous presentation on logic we have primarily discussed propositional calculus. Predicate calculus is a generalized version of propositional calculus. Here, in a predicate statement, a predicate serves a purpose similar to that of a mathematical function and may be applied to either a constant or a variable argument. Hence, a logic statement in this case will take the form:

*Predicate (Argument)*

The predicate describes the object within the argument. For example, the statement

*Is High (Control Gain)*

corresponds to the proposition “*Control gain is high.*” In fact, a more general version would be

*Is High (x).*

Here a variable  $x$  is used as the argument. If  $x$  denotes “control gain” then we have precisely the previous statement. But in a general sense,  $x$  could denote something else like “Temperature.”

Reasoning (or knowledge processing) in predicate calculus is quite similar to, albeit more general than that in, propositional calculus, and may be accomplished through the application of a rule of inference.

#### Example 1.9

Consider the knowledge base consisting of the rule

$$(\forall x) [ \text{Servomotor} (x) \rightarrow \text{Speed-sensor} (x) ]$$

Here  $(\forall x)$  means “*for all x.*” The rule states that all servomotors have speed sensors and may be represented by a model number, which is denoted by the variable  $x$ . Now suppose that a particular positioning system has a servomotor whose model number is H0176. This is now an available data item and is expressed as “*Servomotor (H0176).*” Then by applying the modus ponens rule of inference (MPRI) to the rule base, we obtain the inference “*Speed-sensor (H0176).*” Specifically, this infers that there exists a speed sensor in that particular motor unit.

## 1.5 Soft computing

Soft computing is an important branch of study in the area of intelligent and knowledge-based systems. It has effectively complemented conventional AI in the area of machine intelligence (computational intelligence). Human reasoning is predominantly approximated, qualitative, and “soft.” Humans can effectively handle incomplete, imprecise, and fuzzy information in making intelligent decisions. Fuzzy logic, probability theory, neural networks, and genetic algorithms are cooperatively used in soft computing for knowledge representation and for mimicking the reasoning and decision-making processes of a human. Quite effective are the mixed or hybrid techniques, which synergistically exploit the advantages of two or more of these areas. Decision-making with soft computing involves *approximate reasoning*.

Now we will give an introduction to the subject of soft computing, with an emphasis on fuzzy logic and evolutionary computing or genetic algorithms. The techniques of soft computing will be covered in more detail in other chapters.

### 1.5.1 Fuzzy logic

Fuzzy logic is useful in representing human knowledge in a specific domain of application and in reasoning with that knowledge to make useful inferences or actions. The conventional binary (bivalent) logic is crisp and allows for only two states. This logic cannot handle fuzzy descriptors, examples of which are “fast” which is a *fuzzy quantifier* and “weak” which is a *fuzzy predicate*. They are generally qualitative, descriptive, and subjective and may contain some overlapping degree of a neighboring quantity, for example, some degree of “slowness” in the case of the fuzzy quantity “fast.” Fuzzy logic allows for a realistic extension of binary, crisp logic to qualitative, subjective, and approximate situations, which often exist in problems of intelligent machines where techniques of artificial intelligence are appropriate.

In fuzzy logic, the knowledge base is represented by if-then rules of fuzzy descriptors. An example of a fuzzy rule would be, “If the speed is slow and the target is far, then moderately increase the power,” which contains the fuzzy descriptors *slow*, *far*, and *moderate*. A fuzzy descriptor may be represented by a membership function. This function gives a membership grade between 0 and 1 for each possible value of the fuzzy descriptor it represents.

Mathematically, a fuzzy set  $A$  is represented by a membership function, or a “possibility function” of the form

$$Fz[x \in A] = \mu_A(x): \Re \rightarrow [0, 1] \quad (1.4)$$

in which each element of  $A$ , as denoted by a point  $x$  on the real line  $\Re$ , is mapped to a value  $\mu$  which may lie anywhere in the real interval of 0 to 1. This value is the “grade of membership” of  $x$  in  $A$ . If the membership grade is greater than 0 but less than 1, the membership is not crisp (i.e., it is fuzzy), and the element has some possibility of being within the set and also some

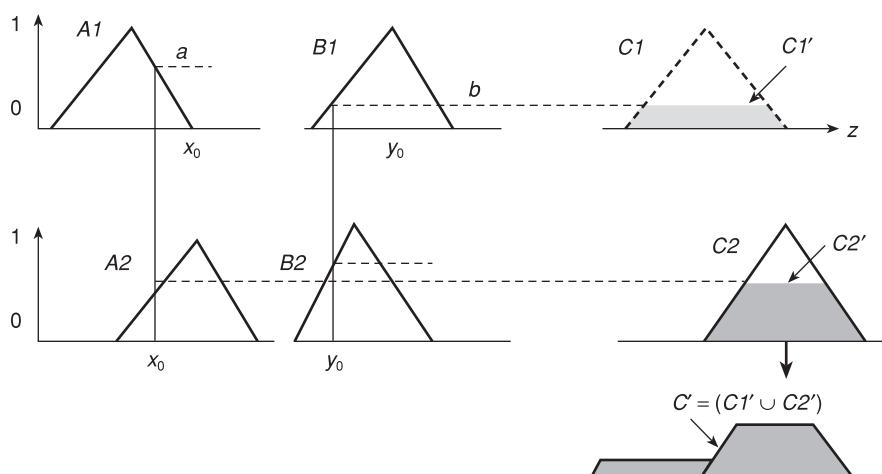
complementary possibility of being outside the set. In that case, the element falls on the fuzzy boundary of the set. The conventional binary (bivalent) logic allows for only two states, represented by 0 and 1, of set membership: an element is either completely inside or completely outside a set. Through the use of membership grades which lie between 0 and 1, fuzzy sets and associated fuzzy logic allow for a realistic extension and generalization of binary, crisp logic.

A fuzzy rule itself may be represented as a grouping of membership functions. An example of two rules:

If  $A1$  and  $B1$  then  $C1$

If  $A2$  and  $B2$  then  $C2$

is sketched in Figure 1.13, where triangular membership functions are used. Here  $A1$  and  $A2$  may represent two fuzzy states such as “near” and “far” of the variable “destination”;  $B1$  and  $B2$  may represent two fuzzy states such as “fast” and “slow” of the variable “speed”; and  $C1$  and  $C2$  may represent the fuzzy actions “decrease” and “increase” of the variable “power.” If the actual distance to the target is  $x_0$  and the actual speed is  $y_0$ , then each fuzzy rule will contribute an action, as shown by the shaded region of the membership functions for  $C1$  and  $C2$  in Figure 1.13. The net power-change action (decision) corresponding to these readings of distance and speed is given by the overall shaded region, indicated by  $C'$  in the figure. This is arrived at according to a particular decision-making procedure – the *sup-min* composition, which is commonly used in fuzzy logic (these concepts will be discussed in detail in a subsequent chapter). If a crisp decision is required, one may use the centroid  $z_0$  of the decision membership function  $C'$  as shown.



**Figure 1.13:** An illustration of fuzzy decision-making

Now consider the general problem of approximate reasoning. In this case the knowledge base  $K$  is represented in an “approximate” form, for example, by a set of if-then rules with *antecedent* and *consequent* variables that are fuzzy descriptors. First, the data  $D$  are preprocessed according to

$$F_D = FP(D) \quad (1.5)$$

which, in a typical situation, corresponds to a data abstraction procedure called “fuzzification” and establishes the membership functions or membership grades that correspond to  $D$ . Then for a fuzzy knowledge base  $F_K$ , the fuzzy inference  $F_I$  is obtained through fuzzy-predicate approximate reasoning, as denoted by

$$F_I = F_K \circ F_D \quad (1.6)$$

Here, the fuzzy matching operator  $FM$  that corresponds to  $M$  in equation (1.1) is in fact the *composition* operator  $\circ$  and may be expressed as

$$\mu_I = \sup_x \min(\mu_K, \mu_D) \quad (1.7)$$

where

$\mu_K$  = multidimensional membership function of the fuzzy rule base

$\mu_D$  = membership function of the fuzzified data

$\mu_I$  = membership function of the fuzzy inference

$x \in X$  = common set of context variables used in knowledge base matching.

Also, the supremum (*sup*) and minimum (*min*) operations are given by the usual definitions (see Chapter 2). Often, a crisp value  $\hat{c}$  is needed for the intelligent action that has to be carried out in the physical process (intelligent machine) and may be determined by the *centroid method*; thus,

$$\hat{c} = \frac{\int_s^c c \mu_I(c) dc}{\int_s^c \mu_I(c) dc} \quad (1.8)$$

where  $c$  is the independent variable of the inference  $I$ , and  $S$  is the support set (or the region of interest) of the inference membership function.

Fuzzy logic is commonly used in direct control of processes and machinery. In this case the inferences of a fuzzy decision-making system form the control inputs to the process. These inferences are arrived at by using the process responses as the inputs (context data) to the fuzzy system.

The simple example of goal pursuit (vehicle driving) shown in Figure 1.13 was given to illustrate the similarity of fuzzy decision-making to approximate reasoning, which is commonly used by humans. One may argue that there is no intelligence in the associated decision-making process because the rules are fixed and no learning and self-improvement are involved. Even in the

example of Figure 1.13, however, there exists a process of “approximation” and the use of approximate reasoning. These, as commonly performed by humans, may be interpreted as manifestations of intelligence. Problems of knowledge-based decision-making that are much more complex would be involved in intelligent machines for practical applications. Use of fuzzy logic is valuable and appealing in such applications, particularly because of its incorporation of approximate reasoning. Learning may be incorporated through a process of reinforcement where valid rules in the rule base are retained and new rules are added (learned), while inappropriate rules are removed. Techniques of neural networks may be used as means of learning in this context. Application areas of fuzzy logic include smart appliances, supervisory control of complex processes, and expert systems.

### 1.5.2 Neural networks

Artificial neural networks (NN) are massively connected networks of computational “neurons,” and represent parallel-distributed processing structures. The inspiration for NN has come from the biological architecture of neurons in the human brain. A key characteristic of neural networks is their ability to approximate arbitrary nonlinear functions. Since machine intelligence involves a special class of highly nonlinear decision-making, neural networks would be effective there. Furthermore, the process of approximation of a nonlinear function (i.e., system identification) by interacting with a system and employing data on its behavior may be interpreted as “learning.” Through the use of neural networks, an intelligent system would be able to learn and perform high-level cognitive tasks. For example, an intelligent system would only need to be presented a goal; it could achieve its objective through continuous interaction with its environment and evaluation of the responses by means of neural networks.

A neural network consists of a set of nodes, usually organized into layers, and connected through weight elements called synapses. At each node, the weighted inputs are summed (aggregated), thresholded, and subjected to an activation function in order to generate the output of that node. These operations are shown in Figure 1.14. The analogy to the operations in a biological neuron is highlighted. Specifically, in a biological neuron, the dendrites receive information from other neurons. The soma (cell body) collects and combines this information, which is transmitted to other neurons using a channel (tubular structure) called an axon. This biological analogy, apart from the abilities to learn by example, approximation of highly nonlinear functions, massive computing power, and memory, may be a root reason for inherent “intelligence” in a neural network. If the weighted sum of the inputs to a node (neuron) exceeds a threshold value  $w_0$ , then the neuron is fired and an output  $y(t)$  is generated according to

$$y(t) = f \left[ \sum_{i=1}^n w_i x_i - w_0 \right] \quad (1.9)$$

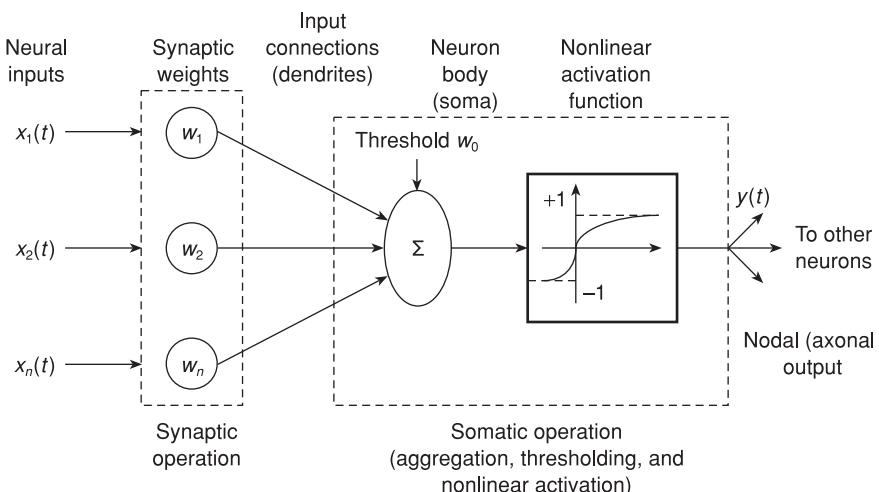


Figure 1.14: The operations at a node of a neural network

where  $x_i$  are neuron inputs,  $w_i$  are the synaptic weights, and  $f[.]$  is the activation function.

There are two main classes of neural networks known as feedforward networks (or static networks) and feedback networks (or recurrent networks). In a feedforward network, an example of which is a multilayer perceptron, the signal flow from a node to another node takes place in the forward direction only. There are no feedback paths. In a feedforward neural network, learning is achieved through example. This is known as supervised learning. Specifically, first a set of input–output data of the actual process is determined (e.g., by measurement). The input data are fed into the NN. The network output is compared with the desired output (experimental data) and the synaptic weights of the NN are adjusted using a gradient (steepest descent) algorithm until the desired output is achieved.

In a feedback NN, the outputs of one or more nodes (say in the output layer) are fed back to one or more nodes in a previous layer (say hidden layer or input layer) or even to the same node. The feedback provides the capability of “memory” to the network. An example is the Hopfield network. It consists of two layers: the input layer and the Hopfield layer. Each node in the input layer is directly connected to only one node in the Hopfield layer. The outputs of the network are fed back to the input nodes via a time delay (providing memory) and synaptic weights. Nodes in the Hopfield layer have nonlinear activation functions such as sigmoidal functions.

Some classes of neural networks use unsupervised learning algorithms. In these learning schemes, the synaptic weights are adjusted based on the input values to the network and the reactions of individual neurons, and not by comparing the network output to the desired output data. Unsupervised learning is called self-organization (or open-loop adaptation) because the output characteristics of the network are determined internally and locally

the network itself, without any data on desired outputs. This type of learning is particularly useful in pattern classification and grouping of data. Hebbian learning and competitive learning are examples of unsupervised learning algorithms. In the Hebbian learning algorithm, the weight between a neuron and an input is strengthened (increased) if the neuron is fired by the input. In competitive learning, weights are modified to enhance a node (neuron) having the largest output. An example is the Kohonen network, which uses a winner-takes-all approach.

### 1.5.3 Genetic algorithms

Genetic algorithms (GA) are derivative-free optimization techniques, which can evolve through procedures analogous to biological evolution. Genetic algorithms belong to the area of evolutionary computing. They represent an optimization approach where a search is made to “evolve” a solution algorithm, which will retain the “most fit” components in a procedure that is analogous to biological evolution through natural selection, crossover, and mutation. In the present context of intelligent machines, intellectual fitness rather than physical fitness is what is important for the evolutionary process. Evolutionary computing can play an important role in the development of an optimal and self-improving intelligent machine.

Evolutionary computing has the following characteristics:

- (1) It is based on multiple searching points or solution candidates (population-based search).
- (2) It uses evolutionary operations such as crossover and mutation.
- (3) It is based on probabilistic operations.

A genetic algorithm works with a population of individuals, each representing a possible solution to a given problem. Each individual is assigned a fitness score according to how good its solution to the problem is. The highly fit (in an intellectual sense) individuals are given opportunities to reproduce by crossbreeding with other individuals in the population. This produces new individuals as offspring, who share some features taken from each parent. The least fit members of the population are less likely to get selected for reproduction and will eventually die out. An entirely new population of possible solutions is produced in this manner, by mating the best individuals (i.e., individuals with best solutions) from the current generation. The new generation will contain a higher proportion of the characteristics possessed by the “fit” members of the previous generation. In this way, over many generations, desirable characteristics are spread throughout the population, while being mixed and exchanged with other desirable characteristics in the process. By favoring the mating of the individuals who are more fit (i.e., who can provide better solutions), the most promising areas of the search space would be exploited. A GA determines the next set of searching points using the fitness values of the current searching points, which are widely distributed

throughout the searching space. It uses the mutation operation to escape from a local minimum.

### 1.5.4 Probabilistic reasoning

Uncertainty and the associated concept of probability are linked to approximation. One can justify that probabilistic reasoning should be treated within the area of soft computing. Probabilistic approximate reasoning may be viewed in an analogous manner to fuzzy logic reasoning, considering uncertainty in place of fuzziness as the concept of approximation that is applicable. Probability distribution/density functions are employed in place of membership functions. The formula of knowledge-based decision-making that corresponds to equation (1.1) or equation (1.6), in this case, depends on the specific type of probabilistic reasoning that is employed. The *Bayesian approach* is commonly used. This may be interpreted as a classification problem.

Suppose that an observation  $d$  is made, and it may belong to one of several classes  $c_i$ . The Bayes' relation states:

$$\max_i P(c_i | d) = \frac{P(d | c_i) \bullet P(c_i)}{P(d)} \quad (1.10)$$

where

$P(c_i | d)$  = given that the observation is  $d$ , the probability that it belongs to class  $c_i$  (the *a posteriori conditional probability*)

$P(d | c_i)$  = given that the observation belongs to the class  $c_i$ , the probability that the observation is  $d$  (the *class conditional probability*)

$P(c_i)$  = the probability that a particular observation belongs to class  $c_i$ , without knowing the observation itself (the *a priori probability*)

$P(d)$  = the probability that the observation is  $d$  without any knowledge of the class.

In the Bayes' decision-making approach, for a given observation (data)  $d$ , *a posteriori* probabilities  $P(c_i | d)$  are computed for all possible classes ( $i = 1, 2, \dots, n$ ), using equation (1.10). The class that corresponds to the largest of these *a posteriori* probability values is chosen as the class of  $d$ , thereby solving the classification problem. The remaining  $n - 1$  *a posteriori* probabilities represent the error in this decision.

Note the analogy between equations (1.10) and (1.6). Specifically,  $P(d)$  represents the “preprocessed” probabilistic data that correspond to the observation  $d$ . The knowledge base itself constitutes the two sets of probabilities:

- (1)  $P(d | c_i)$  of occurrence of data  $d$  if the class (decision) is  $c_i$ ,  $i = 1, 2, \dots, n$
- (2)  $P(c_i)$  of class (decision)  $c_i$ ,  $i = 1, 2, \dots, n$  without any knowledge of the observation (data) itself.

**Table 1.5: Techniques of computational intelligence**

Technique	Characteristic	A popular analogy
Fuzzy logic	Uses fuzzy rules and approximate reasoning	Human knowledge
Neural networks	Network of massively connected nodes	Neuron structure in brain
Genetic algorithms	Derivative-free optimization	Biological evolution
Probability	Incorporates uncertainty in predicting future events	Random action of a human
Conventional AI	Symbolic processing of information	Symbolic languages

The knowledge-base matching is carried out, in this case, by computing the expression on the right side of equation (1.10) for all possible  $i$  and then picking out the maximum value. Application areas of probabilistic decision-making include forecasting, signal analysis and filtering, and parameter estimation and system identification.

Techniques of soft computing are powerful by themselves in achieving the goals of machine intelligence. Furthermore, they have a particular appeal in view of the biological analogies that exist. To summarize the biological analogies of fuzzy, neural, genetic, and probabilistic approaches: fuzzy techniques attempt to approximate human knowledge and the associated reasoning process; neural networks are a simplified representation of the neuron structure of a brain; genetic algorithms follow procedures that are crudely similar to the process of evolution in biological species; and probabilistic techniques can analyze random future action of a human. This is no accident because in machine intelligence it is the behavior of human intelligence that would be mimicked. Popular (e.g., biological) analogies and key characteristics of several techniques of machine intelligence are listed in Table 1.5. Conventional AI, which typically uses symbolic and descriptive representations and procedures for knowledge-based reasoning, is listed as well for completeness.

It is important to compare the concepts of uncertainty and fuzziness. Uncertainty is statistical inexactness due to random future events. Fuzziness arises when the decision of whether a particular object belongs to a given set is a matter of perception, which can be subjective. Probabilistic and statistical techniques may be employed in conjunction with fuzzy logic in a complementary manner. Conceptual differences remain despite the analytical similarities of the two methods. The two concepts are compared in Table 1.6 with respect to their advantages and disadvantages in practical applications.

### 1.5.5 Approximation and intelligence

Approximation is a “soft” concept and is related to intelligence. The capability of approximation for the purpose of comparison, pattern recognition, reasoning and decision-making is a manifestation of intelligence (e.g., dealing with incomplete and subjective information, unfamiliar situations, comparison, and judgment). There are many concepts of approximation, examples

**Table 1.6:** An application-oriented comparison of fuzziness and uncertainty

	Fuzziness	Uncertainty
<b>Advantages</b>	<ul style="list-style-type: none"> <li>■ Incomplete information can be handled</li> <li>■ Particularly useful in representing and processing human-oriented knowledge</li> <li>■ Approximate reasoning is possible, with qualitative and linguistic knowledge</li> <li>■ It is a technique of soft computing</li> </ul>	<ul style="list-style-type: none"> <li>■ Useful in situations having random influences with known probability distributions</li> <li>■ Governs many practical situations</li> <li>■ Mathematical procedures are well established</li> <li>■ System parameters can be determined using crisp experiments</li> </ul>
<b>Disadvantages</b>	<ul style="list-style-type: none"> <li>■ May introduce a degree of inaccuracy</li> <li>■ Needs prior knowledge and experience of the problem in generating the knowledge base</li> <li>■ Can be slow</li> </ul>	<ul style="list-style-type: none"> <li>■ Not related to fuzzy sets</li> <li>■ May fail under incomplete information</li> <li>■ Results are directly affected by the type and accuracy of the probability distributions</li> </ul>

of which include imprecision, uncertainty, fuzziness, and belief. These concepts are not identical even though the associated methods of information representation and processing may display analytical similarities.

Several concepts of approximation that may be useful in knowledge-based reasoning are summarized in Table 1.7. It is important to recognize the characteristic features of the concepts of approximation listed there, so that one could avoid using them incorrectly and inappropriately. In Table 1.7 the main characteristic that differentiates each concept from the others is indicated, and an example to illustrate that characteristic is also given. In representing and analyzing these concepts, one may treat a condition as membership in a set. The set itself may be either fuzzy or crisp depending on the concept. In particular, it should be noted that both uncertainty and imprecision deal with crisp sets and fuzziness uses fuzzy sets. Belief and plausibility are applicable to both crisp and fuzzy sets.

Suppose that the intersection ( $\cap$ ) of fuzzy sets is represented by a norm such as  $\min(x, y)$  and  $xy$  (which are T-norms, as discussed in Chapter 2) and the union ( $\cup$ ) of fuzzy sets is represented by a norm such as  $\max(x, y)$  and  $x + y - xy$  (which are S-norms, as discussed in Chapter 2). Then, for two fuzzy sets  $A$  and  $B$ , the following axiom holds:

$$Fz(A \cup B) = Fz(A) + Fz(B) - Fz(A \cap B) \quad (1.11)$$

in which  $Fz()$  denotes the grade of membership of an element  $x$  in the set that is specified within ( ).

The concept of approximation given by “uncertainty” deals with the “probability of membership” in a crisp set, not a fuzzy set. In this case, a probability function  $p(x)$  may be defined for a crisp set  $A$  as follows:

**Table 1.7:** Several concepts of approximation and their characteristics

Concept	Property	Example
Ambiguity	The condition has several different possibilities, and it is not determined which one is valid.	The machine response “may or may not” satisfy the specification.
Vagueness	The condition is not precisely (clearly) defined.	The machine response “may have” met the specification.
Generality	The condition may apply to many (finite or infinite) situations depending on the specific context.	The machine response is “ $x$ ” times the specification.
Imprecision	Condition can assume a state within a clearly defined (crisp) tolerance interval.	The machine response is “within $\pm 5\%$ ” of the specification.
Uncertainty	There is a degree of probability associated with occurrence of the condition.	There is a “90% probability” that the machine response meets the specification.
Fuzziness	The membership of the condition is not crisply defined (set boundary of the condition is not crisp).	The machine response is “close to” the specification.
Belief (Subjective Probability)	The level of belief on the condition (membership of a crisp or fuzzy set) is through knowledge and evidence.	It is believed at a level of 90% that the machine response meets the specification.
Plausibility	The plausibility of nonmembership fully complements the belief of membership (dual condition of belief). $Bl(x \in A) + Pl(x \notin A) = 1$	It is plausible at a level of 95% that the machine response meets the specification.

$$Pr[x \in A] = p(x): \Re \rightarrow [0, 1] \quad (1.12)$$

The axiom that describes the probability of combined sets is analogous to equation (1.11). Specifically,

$$Pr(A \cup B) = Pr(A) + Pr(B) - Pr(A \cap B) \quad (1.13)$$

in which  $A$  and  $B$  are crisp (non-fuzzy) sets, and  $Pr(S)$  is interpreted as the probability that an element  $x$  belongs to set  $S$ . From equations (1.4) and (1.11) through (1.13), some form of analytical similarity between fuzziness and uncertainty may be established. But clearly these two concepts of approximation are not identical. Nevertheless, probabilistic and statistical techniques may be employed in fuzzy logic, and similarly, probabilistic approaches may be augmented with fuzzy logic techniques. Specifically, a membership function of a fuzzy set may be estimated by using statistical data on membership and nonmembership of the set, as determined by, say, polling a set of experts. For example, a set of trials may be conducted among a group of experts to determine whether a particular state is “fast” or not. Then the grade of membership may be estimated as the ratio of the number of positive

polls to the total number of polls. An entire membership function can be estimated by repeating this experiment for different conditions of speed. This practical link between fuzziness and uncertainty may be extended to establish an analytical link as well, while giving due consideration to the dissimilarity between the two concepts. To further establish a link of this form, another interpretation for fuzziness, that is somewhat akin to probability, is also available. Here, a crisp set  $A$  is considered. Then, the level of fuzziness of an element  $x$  is interpreted as the truth value, or the level of belief, or the level of certainty that  $x \in A$ . Another common way of jointly exploiting the concepts of fuzziness and uncertainty is through the use of *fuzzy modifiers*. An example of a fuzzy modifier would be “fast with a certainty level of 90%.”

Yet another useful concept of approximation is *belief*, as listed in Table 1.7. Here,  $Bl(A)$  denotes the level of belief that the element  $x$  belongs to set  $A$ . A belief value may represent the subjective level of confidence (or belief) that a particular object is a member of a given set and may be based on available knowledge, experience, common sense, experimentation, and other evidence. A belief may be defined regardless of whether the set is crisp or fuzzy. The axiom of combination for belief is

$$Bl(A \cup B) \geq Bl(A) + Bl(B) - Bl(A \cap B) \quad (1.14)$$

with

$$Bl(X) = 1; \quad Bl(\emptyset) = 0 \quad (1.15)$$

in which  $X$  denotes the universal set and  $\emptyset$  denotes the null set. When  $A$  and  $B$  are disjoint sets, one has

$$Bl(A \cup B) \geq Bl(A) + Bl(B) \quad (1.16)$$

It follows that the level of belief can improve when two sets are considered jointly within a common context over when they are considered separately and independently. Also, it follows from (1.16) that

$$1 \geq Bl(A) + Bl(A') \quad (1.17)$$

where  $A'$  is the complement (negation) of  $A$ .

Expression (1.17) states that the belief of a nonoccurrence of  $A$  is less than the complement of belief of  $A$ . In other words, the belief of occurrence of  $A$  and the belief of nonoccurrence of  $A$  are not full, and there exists some level of belief that is indecisive. This deficiency is compensated for by introducing the concept of plausibility, which satisfies

$$Pl(A') = 1 - Bl(A) \quad (1.18)$$

From equation (1.18) it also holds that

$$Pl(A) = 1 - Bl(A') \quad (1.19)$$

Furthermore, it can be shown that

$$Pl(A) \geq Bl(A) \quad (1.20)$$

and

$$Pl(A') \geq Bl(A') \quad (1.21)$$

In view of the inequalities (1.20) and (1.21) one could argue that, on the basis of a given body of knowledge (including evidence, experience, and common sense), the level of plausibility that a particular situation occurs is greater than the level of belief of occurrence of the situation. Hence, plausibility is a more liberal, weaker, or more flexible form of belief. As is the case with belief, plausibility may be applicable to both crisp sets and fuzzy sets. The complement relationship for plausibility is given by

$$1 \leq Pl(A) + Pl(A') \quad (1.22)$$

which further supports the weaker or more liberal nature of plausibility over belief (inequality (1.17)).

Some useful properties of four common concepts of approximation are summarized in Table 1.8. Note that all these concepts can be described in terms of an “inexactness” of membership in a set. The nature and origin of the inexactness, however, are unique to the particular concept itself. The type of set used is also not the same for all four concepts. Specifically, uncertainty concerns crisp sets and fuzziness deals with fuzzy sets, whereas belief and plausibility are applicable to both crisp and fuzzy sets. All four concepts are useful in knowledge-based decision-making of intelligent systems, where the representing and processing of knowledge are crucial. Even though the

**Table 1.8: A comparison of several concepts of approximation**

Concept	Qualitative definition	Combination axiom	Complement relation
Uncertainty	Probabilistic/statistical inexactness of occurrence of an event (crisp set)	$Pr(A \cup B) = Pr(A) + Pr(B) - Pr(A \cap B)$	$Pr(A) + Pr(A') = 1$
Fuzziness	Inexactness in human perception about a situation (set boundary is non-crisp)	$Fz(A \cup B) = Fz(A) + Fz(B) - Fz(A \cap B)$	$Fz(A) + Fz(A') = 1$
Belief	Inexactness in human belief or confidence about a condition (crisp or fuzzy set)	$Bl(A \cup B) \geq Bl(A) + Bl(B) - Bl(A \cap B)$	$Bl(A) + Bl(A') \leq 1$
Plausibility	A more liberal form of belief. Level is higher than belief, with the same amount of evidence (crisp or fuzzy set)	$Pl(A \cup B) \leq Pl(A) + Pl(B) - Pl(A \cap B)$	$Pl(A) + Pl(A') \geq 1$

mathematical formulations are somewhat similar and the physical characteristics would be lost in the procedures of numerical computations, it is important to understand the differences in meaning of these concepts when employing them in practical applications.

It is possible to incorporate and employ these various concepts of approximation into a knowledge base in a complementary, rather than mutually exclusive, manner. For example, analogous to how fuzziness and uncertainty are jointly applied to a condition by the use of fuzzy modifiers, fuzziness may be applied jointly with either belief or plausibility. In this context *fuzzy beliefs* or *fuzzy plausibilities*, which are akin to fuzzy truth values, may be employed. This combination of approximators may be achieved, for example, by using expressions of the form “fast with a 90% level of belief.”

Uncertainty and belief levels may be included as qualifiers in a rule, for example,

If  $A$  then  $B$  with probability  $p$ .

If  $A$  then  $B$  with belief  $b$ .

In each case, the antecedent  $A$  and the consequent  $B$  can be either crisp or non-crisp (fuzzy) quantities. In the fuzzy case the level of fuzziness of a parameter is determined by the degree of membership of that parameter in the associated fuzzy set. Apart from that, a level of certainty  $p$ , a truth value  $m$ , a level of belief  $b$ , etc. may be associated with each rule, so as to qualify the exactness or confidence of each rule on the basis of additional statistical information, perception, evidence, and so on. In this manner, the concepts of approximation may be used in a complementary way to more realistically and authentically represent a knowledge base and to make valid inferences using it.

Formulas that correspond to equation (1.1) or equation (1.6) may be identified for knowledge-based decision-making involving other concepts of approximation (e.g., belief, plausibility, truth value) as well. For example, in the case of belief, the knowledge base may constitute a set of belief values for a set of rules. Then, knowing the belief value of an observation, the decision-making problem will become a matter of picking the segment of the rule base that matches the data and also has the highest belief. More often, however, such concepts of approximation are used as modifiers within a knowledge base involving a different concept of approximation (e.g., fuzzy or probabilistic) rather than stand-alone tools of decision-making, as alluded to previously.

### 1.5.6 Technology needs

Even though a considerable effort has gone into the development of systems and machines that somewhat mimic humans in their actions, the present generation of intelligent systems do not claim to possess all the capabilities of human intelligence – for example, common sense, display of emotions, and inventiveness. Significant advances have been made, however, in machine implementation of characteristics of intelligence such as sensory perception,

pattern recognition, knowledge acquisition and learning, inference from incomplete information, inference from qualitative or approximate information, ability to handle unfamiliar situations, adaptability to deal with new yet related situations, and inductive reasoning. Much research and development would be needed in these areas, pertaining to techniques, hardware, and software, before a machine could reliably and consistently possess the level of intelligence of, say, a dog.

### Example 1.10

Consider a handwriting recognition system, which is a practical example of an intelligent system. The underlying problem cannot be solved through simple template matching, which does not require intelligence. Handwriting of the same person can vary temporally, due to various practical shortcomings such as missing characters, errors, non-repeatability, physiological variations, sensory limitations, and noise. It should be clear from this observation that a handwriting recognition system has to deal with incomplete information and unfamiliar objects (characters), and should possess capabilities of learning, pattern recognition, and approximate reasoning, which will assist in carrying out intelligent functions of the system. Techniques of soft computing are able to challenge such needs of intelligent machines.

## 1.6 Summary

This chapter provides a general introduction to the subjects of machine intelligence, knowledge-based systems, and soft computing, and forms a common foundation for the remaining chapters of the book. The concept of artificial intelligence was critically discussed and the linkage between intelligence and approximation was examined. Several concepts of approximation were presented and an analytical basis for them was given. Knowledge-based intelligent systems were discussed in some detail. Their architectures and methods of representing and processing of knowledge were examined. Expert systems, which represent a practical class of knowledge-based systems, were studied. Techniques of soft computing, particularly fuzzy logic, neural networks, genetic algorithms, and probabilistic reasoning, were outlined. The presentation given in this chapter is intended to be neither exhaustive nor highly mathematical. A more rigorous treatment of some of the topics introduced here is found in other chapters.

### Problems

1. If " $A \rightarrow B$ " is  $F$  and " $B \rightarrow A$ " is  $T$ , what is the truth value of " $(A \rightarrow B) \text{ AND } (B \rightarrow A)$ "? Using this result and the truth table of " $A \rightarrow B$ ", determine the truth table of " $A \leftrightarrow B$ ".

2. Consider the following knowledge base:

*The step response of the robot has large oscillations.*

*If a plant response is oscillatory then increase the derivative action.*

What rule of inference may be used in processing this and what is the resulting inference? Indicate the steps involved in the reasoning process.

3. Consider the knowledge base:

*LOW means less than 16°C.*

*SLIGHT means 1 division.*

*If the temperature is LOW then SLIGHTLY increase the thermostat setting.*

Is this a fuzzy knowledge base? Explain.

4. Which method of knowledge representation and processing given below would be the most appropriate model for human reasoning?

(a) Two-valued logic  
(b) Semantic networks

(c) Frames  
(d) Production systems.

5. Define the term “information resolution.”

In a hierarchical control system, the degree of intelligence needed for various actions generally increases with the hierarchical level. Also, the information resolution generally decreases with the hierarchical level. Considering the example of a manufacturing workcell, verify these relationships.

6. In conventional computer programs, data, instructions, and reasoning are all integrated. But knowledge-based application programs are usually “object-oriented” where these three items are separately structured and located. Give an advantage of this separation.

7. One definition of an expert system is “a computer program that embodies expert knowledge and understanding about a particular field of expertise, which can be used to assist in the solution of a problem in that field.” List several areas where expert systems are found useful. Choose a specific application and develop a simple rule base that may be used in a knowledge-based system.

8. In statistical process control (SPC), decisions as to whether a process is in control or not are made using a control chart. A control chart consists simply of two lines, the *lower control line* and the *upper control line*, drawn parallel to the time axis, and are in units of the controlled (response) variable, as shown in the Figure P1.8.

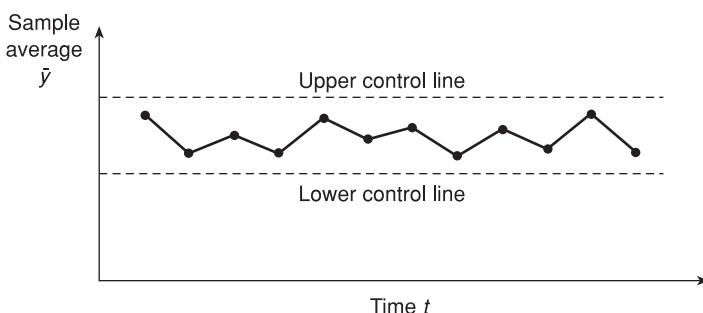


Figure P1.8: A control chart

During operation of the plant, the variable (response) of interest is sampled at fixed intervals, and a sample average is computed. If this value remains within the two control lines, the process is in control. In this case the process error that exists is caused by inherent random causes and can be neglected. If the control limits are exceeded, appropriate control actions are taken to correct the situation. Do you consider SPC an intelligent control technique? Explain.

- 9.** Suppose that a nonconventional, “intelligent” controller generates the following control inferences:

- (a) Increase the control signal to some value.
- (b) You may or may not increase the control signal.
- (c) Increase the control signal to value (variable)  $x$ .
- (d) Increase the control signal to 2.5 within a tolerance of  $\pm 5\%$ .
- (e) Increase the control signal to 2.5 with a probability of 90%.
- (f) Increase the control signal slightly.

Indicate which of these inferences are (i) vague, (ii) ambiguous, (iii) general, (iv) imprecise, (v) uncertain, and (vi) fuzzy (i.e., non-crisp).

- 10.** Expert systems should be able not only to provide answers to questions but also to provide explanations for, or reasoning behind, these answers. Furthermore, they should have the capability to consider alternative goals, not just the single goal that is implied by a specific question. For example, suppose that an expert system for condition monitoring of a workcell is asked the question, “Is the performance acceptable?” With *backward chaining*, a response of “Yes” might be generated. This response might be true but would not be adequate. Explore this, and indicate what other details could be expected from the expert system.
- 11.** A difficult and important task in the development of any knowledge-based (or artificial-intelligence) application is the process of *knowledge engineering*. This concerns the acquisition of expert knowledge and representation of the knowledge in a form that is compatible with the specific knowledge-based system. What are some other important considerations in developing such an application?
- 12.** Critically evaluate the statement “power of an expert system is derived from its knowledge base, not from the particular formalism and inference scheme it employs.”
- 13.** The decision tree shown in Figure P1.13 provides a simple knowledge base for climate control of a room. Translate this into a rule base.

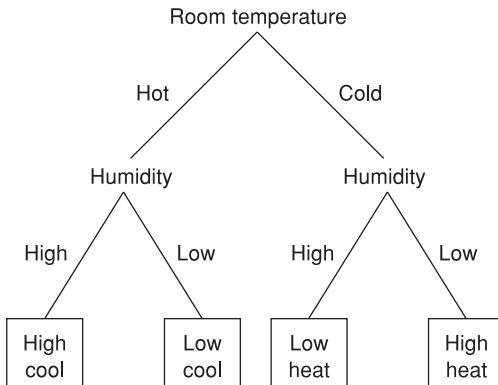


Figure P1.13: The decision tree of a climate control system

14. What is an intelligent machine? Describe a practical example of an intelligent system. Discuss the attributes that are commonly exhibited by an intelligent system that make it intelligent, and explain why these attributes make the system intelligent.
15. Discuss relevant stages of development of an intelligent product or system from its conception to widespread application and usage. What are obstacles that need to be overcome in each of these stages? Suggest ways to overcome these obstacles.
16. What is an expert system? What are performance goals of the next generation expert systems? Discuss whether fuzzy logic is appropriate for use in an expert system.
17. Critically compare and contrast “fuzziness” and uncertainty. In this comparison, specifically address the following issues, pertaining to each of these two concepts:
  - (a) The concept and its meaning.
  - (b) The mathematical representation and analysis.
  - (c) Appropriate application areas.
  - (d) Advantages and disadvantages.

Discuss how we might incorporate (combine) these two concepts in a single application. What is the rationale (justification) for such a combined use? Give a practical example where the combined use of fuzziness and uncertainty would be important.

18. What is soft computing? Indicate biological analogies of the basic techniques of soft computing. Describe why soft computing is particularly useful in representing and reasoning with human-originated knowledge.
19. In what type of knowledge-based applications are the techniques of fuzzy logic particularly appropriate? These techniques can be strengthened and enhanced by using techniques of neural networks, evolutionary computing, and probability. Explain the reasons for this fact.
20. The ability to approximate is a characteristic of an intelligent system. Giving examples, justify this statement. How closely do the most sophisticated modern applications of machine intelligence resemble human intelligence?

## References

1. Anderson, T.W. (1984) *An Introduction to Multivariate Statistical Analysis*, John Wiley & Sons, New York.
2. Davis, L. (1991) *Handbook of Genetic Algorithms*, Van Nostrand Rienhold, New York.
3. De Silva, C.W., and Lee, T.H. (1994) “Knowledge-based intelligent control,” *Measurements and Control*, vol. 28, no. 2, pp. 102–113, April.
4. De Silva, C.W. (1995) *Intelligent Control: Fuzzy Logic Applications*, CRC Press, Boca Raton, FL.
5. De Silva, C.W. (1997) “Intelligent control of robotic systems with application in industrial processes,” *Robotics and Autonomous Systems*, vol. 21, pp. 221–237.

6. De Silva, C.W. (ed.) (2000) *Intelligent Machines – Myths and Realities*, CRC Press, Boca Raton, FL.
7. De Silva, C.W. (2003) “The role of soft computing in intelligent machines,” *Philosophical Transactions of the Royal Society*, Series A, UK.
8. Filippidis, A., Jain, L.C., and De Silva, C.W. (1999) “Intelligent control techniques,” *Intelligent Adaptive Control* (editors: Jain, L.C. and De Silva, C.W.), CRC Press, Boca Raton, FL.
9. Gupta, M.M., and Rao, H. (1994) *Neural Control Theory and Applications*, IEEE Press, Piscataway, NJ.
10. Hart, A. (1986) *Knowledge Acquisition for Expert Systems*, McGraw-Hill, New York.
11. Lee, T.H., Yue, P.K., and De Silva, C.W. (1994) “Neural networks improve control,” *Measurements and Control*, vol. 28, no. 4, pp. 148–153.
12. Mongi, A.A., and Gonzalez, R.C. (1992) *Data Fusion in Robotics and Machine Intelligence*, Academic Press, Orlando, FL.
13. Pao, Y.H. (1989) *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley, Reading, MA.
14. Shafer, G. (1976) *A Mathematical Theory of Evidence*, Princeton University Press, Princeton, NJ.
15. Staugaard, A.C. (1987) *Robotics and AI*, Prentice Hall, Inc., Englewood Cliffs, NJ.
16. Zadeh, L.A. (1984) “Making computers think like people,” *IEEE Spectrum*, pp. 26–32, August.



# Fundamentals of fuzzy logic systems

## Chapter outline

2.1	Introduction	57
2.2	Background	58
2.3	Fuzzy sets	65
2.4	Fuzzy logic operations	68
2.5	Generalized fuzzy operations	76
2.6	Implication (if-then)	82
2.7	Some definitions	89
2.8	Fuzziness and fuzzy resolution	91
2.9	Fuzzy relations	97
2.10	Composition and inference	105
2.11	Considerations of fuzzy decision-making	126
	Problems	128
	References	135

## 2.1 Introduction

Fuzzy logic was first developed by L.A. Zadeh in the mid-1960s for representing some types of “approximate” knowledge that cannot be represented by conventional, crisp methods. In the crisp Boolean logic, truth is represented by the state 1 and falsity by the state 0. Boolean algebra and crisp logic have no provision for approximate reasoning. Fuzzy logic is an extension of crisp bivalent (two-state) logic in the sense that it provides a platform for handling approximate knowledge. Fuzzy logic is based on fuzzy set theory in a similar manner to how crisp bivalent logic is based on crisp set theory. A fuzzy set is represented by a *membership function*. A particular “element” value in the range of definition of the fuzzy set will have a grade of membership which gives the degree to which the particular element belongs to the set. In this manner it is possible for an element to belong to the set at

some degree and simultaneously not belong to the set at a complementary degree, thereby allowing a non-crisp (or fuzzy) membership. This chapter presents the basic theory of fuzzy logic, which uses the concept of fuzzy sets. Applications of soft computing and particularly fuzzy knowledge-based systems rely on the representation and processing of knowledge using fuzzy logic. In particular, the *compositional rule of inference* is what is applied in decision-making with fuzzy logic. Underlying principles are systematically presented in this chapter. Geometrical illustrations and examples are added in order to facilitate the interpretation. The application of these concepts in knowledge-based “intelligent” systems and particularly in intelligent control will be treated in future chapters.

## 2.2 Background

Humans are flexible (or soft) systems. They can adapt to unfamiliar situations, and they are able to gather information in an efficient manner and discard irrelevant details. The information that is gathered need not be complete and precise and may be *general*, *qualitative*, *approximate*, and *vague*, for humans can *reason*, *infer*, and *deduce* new information and knowledge. They have common sense. They can make good decisions, and also can provide logical explanations for those decisions. They can *learn*, *perceive*, and improve their skills through experience. Humans can be creative, inventive, and innovative. It is a very challenging task to seek to develop systems that possess even a few of these simple human abilities. This is the challenge faced by workers in the field of artificial intelligence (AI) and *knowledge-based intelligent systems*. However, humans have weaknesses too. For example, they tend to be slow, inaccurate, forgetful, and emotional. In some other situations, one might argue that these human qualities are strengths rather than weaknesses.

In a knowledge-based intelligent system, one seeks to combine the advantages of a computer with some of the intelligent characteristics of a human for making *inferences*, useful decisions, and actions. It is clearly not advisable to attempt simply to mimic human thought process without carefully studying the needs of the problem at hand. In fact a faithful duplication is hardly feasible. Rather, in the initial stages of development of this field, it is adequate to develop special-purpose systems that cater for a limited class of problems. Knowledge-based intelligent control is one such development.

Knowledge-based systems rely on some method for representing and processing of knowledge. One approach to representing and processing of knowledge is logic. Two-state (bivalent), crisp logic uses only two quantities, *true* (T) and *false* (F) as truth values. Real-life situations do not always assume two crisp states T and F with a crisp line dividing them. Also, linguistic descriptors such as “fast,” “warm,” and “large” are not crisp quantities and tend to be quite subjective, approximate, and qualitative. Then, a statement such as “The water is warm” may have a truth value that is partially true and partially false. Conventional, bivalent logic is inadequate to handle such

situations. Conventional logic introduces some well-known paradoxes as well. For example, the statement “All my statements are lies” does not have a crisp truth value, and hence is not amenable to conventional, two-state logic. This statement, if assumed to be true, will be false, and vice versa. Fuzzy logic is a quite general form of logic that can deal with many such non-crisp situations.

### Example 2.1

An “Iron Butcher” is a machine that is commonly used for cutting off the head of a fish, which is the first stage in the fish canning process. The annual wastage of useful salmon due to inaccurate head cutting using these somewhat outdated machines is estimated at millions of dollars for the fish processing industry. The main cause of wastage is the “over-feed problem.” This occurs when a salmon is inaccurately positioned with respect to the cutter blade so that the cutting location is beyond the collar bone and into the body of a salmon. Similarly, an “under-feed” problem occurs when the cutting is incomplete, and unpalatable parts of the fish head are retained with the processed fish body. An effort has been made to correct this situation by sensing the position of the collar bone using a vision system and automatically positioning the cutter blade accordingly. The Intelligent Iron Butcher has been developed in the Industrial Automation Laboratory of the University of British Columbia, Canada, to automate the fish cutting process. The goal is to realize fast and accurate cuts so as to minimize the amount of useful meat that is wasted and to increase the throughput.

The machine operates using an AC motor-powered reciprocating indexer to accurately move the fish from the feeder end to the vision modules and the cutter assembly of the machine. The cutter assembly consists of a two-axis orthogonal electro-hydraulic planar positioning system (manipulator) on which a pneumatically operated heavy-duty guillotine blade is mounted. There are two vision modules, each consisting of a CCD camera, frame grabber, and an image processor. Prior to cutting, the first vision module is used to analyze the geometry of each fish, particularly the position of the gill with respect to the reference frame of the cutter module, in order to determine the correct x-y coordinates for moving the blade. The motion command is streamed to the hydraulic manipulator in the form of a step input for each cut. Position control of the cutter and the speed control of the conveyor are accomplished using conventional means, particularly through servo control. The second vision module is located beyond the cutter assembly. This module inspects the quality of cut, feeds in the information to the hierarchical supervisory controller of the machine, and on that basis several control adjustments are made, particularly for tuning and performance enhancement of the machine. These adjustments may include descriptive (e.g., small, medium, large) changes in the cutter blade speed, conveyor speed, and the holding force. The required degree of adjustment is learned through experience. It is not generally possible to establish

crisp control algorithms for such adjustments, and the control knowledge may be expressed as a set of rules such as:

*If the “quality” of the processed fish is “not acceptable”, and if the cutting load appears to be “high”, then “moderately” decrease the conveyor speed, or else if the hold-down force appears to be “low”, then “slightly” increase the holding force, or else .....*

Such a set of rules forms the *knowledge base* of this control problem. It should be clear that this rule base has *qualitative*, *descriptive* and *linguistic* terms such as “quality”, “acceptable”, “high”, and “slight”. These are fuzzy descriptors. Crisp representations and algorithms used in conventional control cannot directly incorporate such knowledge. Fuzzy logic control is used in the high-level control of this machine.

Fuzzy logic provides an approximate yet practical means of representing knowledge regarding a system (e.g., describing the behavior of the system) that is too complex or ill-defined and not easy to tackle using precise mathematical means. The approach also provides a means of making inferences using that knowledge, which can be used in making correct decisions regarding the system and for carrying out appropriate actions. In particular, human-originated knowledge can be effectively handled using fuzzy logic. As the complexity of a system increases, the ability to develop precise analytical models of the system diminishes until a threshold is reached beyond which analytical modeling becomes intractable. Under such circumstances, precise model-based decision-making is not practical. Fuzzy knowledge-based decision-making is particularly suitable then.

### 2.2.1 Evolution of fuzzy logic

Early work on fuzzy sets and fuzzy logic originated in the 1960s in the USA, with the seminal work of Zadeh. Subsequent developments and industrial applications in Europe and other regions in the 1970s, and the widespread commercial application in consumer appliances, ground transportation, and various industrial products, primarily in Japan in the 1980s, have established this field with the respect it rightfully deserves. The October 19, 1987 issue of *Nikkei Industrial News* stated: “Toshiba has developed an AI system which controls machinery and tools using Fuzzy Logic. It controls rules, simulation, and valuation. Toshiba will add Expert System function to it and accomplish the synthetic AI. Toshiba is going to turn it into practical uses in the field of industrial products, traffic control, and nuclear energy.” This news item is somewhat ironic and significant, because around the same time at the Information Engineering Division of the University of Cambridge a similar application of fuzzy logic was developed for a robot. This work was subsequently extended in the Industrial Automation Laboratory of the University

of British Columbia, where the applications centered around the fish processing industry. Many engineers, scientists, researchers, and other professionals throughout the world have made significant contributions in bringing the field of fuzzy logic into maturity, both in research and practical applications. These contributions are too numerous to mention here, but some popular applications are listed below. The information is based on various sources, and may not be exact.

#### 2.2.1.1 Popular applications

**Process temperature control (OMRON):** This is a proportional-integral-derivative (PID) controller integrated with a fuzzy controller. When temperature control is initiated, only the PID action is present. If the temperature overshoots the set point, the fuzzy controller takes over. This control system responds well to disturbance.

**Air conditioner (Mitsubishi):** Conventional air conditioning systems use on-off controllers. When the temperature drops below a preset level the unit is automatically turned off. When the temperature rises above a preset level the unit is turned on. The former preset value is slightly lower than the latter preset value, providing a dead zone, so that high-frequency on-off cycling (chatter) is avoided. The thermostat in the system controls the on-off action. For example, “when the temperature rises to 25°C, turn on the unit, and when the temperature falls to 20°C, turn off the unit.” The Mitsubishi air conditioner controls by using fuzzy rules such as: “If the ambient air is getting warmer, turn the cooling power up a little; if the air is getting chilly, turn the power down moderately, etc.” The machine becomes smoother as a result. This means less wear and tear of the air conditioner, more consistent comfortable room temperatures, and increased efficiency (energy savings).

**Vacuum cleaner (Panasonic):** Characteristics of the floor and the amount of dust are sensed by an infrared sensor, and the microprocessor selects the appropriate power by fuzzy control according to these characteristics. The floor characteristics include the type (hardwood, cement, tile, carpet softness, carpet thickness, etc.). The changing pattern of the amount of dust passing through the infrared sensor is established as well. The microprocessor establishes the appropriate setting of the vacuum head and the power of the motor, using a fuzzy control scheme. Red and green lamps of the vacuum cleaner show the amount of dust left on the floor.

**Automatic transmission system (Nissan, Subaru, Mitsubishi):** In a conventional automatic transmission system, electronic sensors measure the vehicle speed and throttle opening, and gears are shifted based on the predetermined values of these variables. According to Nissan, this type of system is incapable of uniformly providing satisfactory control performance to a driver because it provides only about three different shift patterns. The fuzzy control transmission senses several variables including vehicle speed and acceleration,

throttle opening, the rate of change of throttle opening, engine load, and driving style. Each sensed value is given a weight, and a fuzzy aggregate is calculated to decide whether to shift gears. This controller is said to be more flexible, smooth, and efficient, providing better performance. Also, an integrated system developed by Mitsubishi uses fuzzy logic for active control of the suspension system, four-wheel-drive (traction), steering, and air conditioning.

**Washing machine (Matsushita, Hitachi):** The control system senses both quality and quantity of dirt, load size, and fabric type, and adjusts the washing cycle and detergent amount accordingly. Clarity of water in the washing machine is measured by light sensors. At the start of the cycle, dirt from clothes will not have yet reached the water, so light will pass through it easily. The water becomes more discolored as the wash cycle proceeds, and less light will pass through. This information is analyzed and control decisions are made using fuzzy logic.

**Camcorder (Panasonic, Sanyo, Fisher, Canon):** The video camera determines the best focus and lighting, particularly when several objects are in the picture. Also, it has a digital image stabilizer to remove hand jitter. Fuzzy decision-making is used in these actions. For example, the following scheme is used for image stabilization. The present image frame is compared with the previous frame from memory. A typically stationary object (e.g., house) is identified and its shift coordinates are computed. This shift is subtracted from the image to compensate for the hand jitter. A fuzzy algorithm provides a smooth control/compensation action.

**Elevator control (Fujitec, Toshiba):** A fuzzy scheme evaluates passenger traffic and the elevator variables (load, speed, etc.) to determine car announcement and stopping time. This reduces waiting time and improves the efficiency and reliability of operation.

**Handheld computer (Sony):** A fuzzy logic scheme reads the hand-written input and interprets the characters for data entry.

**Television (Sony):** A fuzzy logic scheme uses sensed variables such as ambient lighting, time of day, and user profile, and adjusts such parameters as screen brightness, color, contrast, and sound.

**Antilock braking system (Nissan):** The system senses wheel speed, road conditions, and driving pattern, and the fuzzy ABS determines the braking action, with skid control.

**Subway train (Sendai):** A fuzzy decision scheme is used by the subway trains in Sendai, Japan, to determine the speed and stopping routine. Ride comfort and safety are used as performance requirements.

Other applications of fuzzy logic include a hot water heater (Matsushita), a rice cooker (Hitachi), and a cement kiln (Denmark). A fuzzy stock-trading

program can manage stock portfolios. A fuzzy golf diagnostic system is able to select the best golf club based on size, characteristics, and swing of a golfer. A fuzzy mug search system helps in criminal investigations by analyzing mug shots (photos of the suspects) along with other input data (say, statements such as “short, heavy-set, and young-looking . . .” from witnesses) to determine the most likely criminal. Gift-wrapped chocolates with fuzzy statements are available for Valentine’s Day. Even a Yamaha “fuzzy” scooter was spotted in Taipei.

### 2.2.2 Stages of development of an intelligent product

Typical stages of development of an “intelligent” product are outlined now, enabling us to understand the slow acceptance of fuzzy logic and Japan’s dominance in fuzzy logic applications.

- (1) Conception of the idea of the product and discussion with interested parties for product development. Obstacles to this first stage include limited awareness of the required technologies. Managers and even engineers within a particular company or group may not have sufficient knowledge of soft computing, AI, knowledge-based systems, and advanced sensor technologies, and their advantages.
- (2) Demonstration of the concept using simplified examples. This involves innovation. The main obstacle here is possible skepticism of co-workers, managers, potential sponsors, and users. Expectations of managers, investors, etc., will be much greater than what the product could deliver in practice. Feasibility demonstrations by building a small prototype system to demonstrate that a knowledge system would actually be useful in a given context can remove this obstacle to some extent.
- (3) Development of practical specifications and designing a product prototype to satisfy the specifications. The main obstacles in this stage include shortage of technical personnel (e.g., domain experts, knowledge engineers), tools, funds, and other resources and infrastructure. In particular, novel applications may not be amenable to existing knowledge-representation and processing (knowledge-based system) techniques and architectures. Often, equipment in research and development (R&D) laboratories may have to be modified or even new devices may have to be developed for prototype design and manufacture. The use of flexible, portable, and commonly available tools and techniques is desirable here.
- (4) Integration of the product into an existing system. This is primarily a problem of technology transfer. Corporate inertia, established work structures and practices, and worker fears of new technology may present obstacles here. Simplified and user-friendly system interfaces, training and re-training of workers for the new technology, nature of the workforce, continuous interaction between various groups (R&D personnel, managers, operators, maintenance personnel, etc.), and due respect for existing corporate structures and practices will help here.

A new product will change the state of the art. When introducing it to the society at large, some of the same considerations as those mentioned under technology transfer should be addressed. Ready acceptance of a new product may be difficult due to such factors as cultural inertia, fear of advanced technologies, complexity, and consumer habits. Proper marketing, user-friendly manuals, and financial incentives will help in this regard.

Slow acceptance of fuzzy logic, particularly in the western world, may be attributed to:

- Negative connotation of the term “fuzzy” in the society at large
- Simplicity of application of fuzzy logic, which could be incorrectly interpreted by some as lack of sophistication and analytical foundation
- Difficulty of changing well-established beliefs, ways, and practices in the research, education, and industrial communities
- Lack of sufficient exposure (media, journals, conferences, etc.)
- Lack of useful demonstration of the technology, in the earlier period of its development.

Reasons for Japan’s dominance in fuzzy logic applications include:

- Cultural religious use and wide acceptance of fuzzy concepts. Directness of expression may not be regarded as a desirable attribute
- Far greater tolerance and accommodation for inexactness and ambiguity
- Mass production of consumer appliances enabled a simple and cost-effective incorporation of “intelligence” into these appliances using fuzzy logic
- Consumer preference for “intelligent” products and willingness to pay more for them.

### 2.2.3 Use of fuzzy logic in expert systems

Fuzzy logic is particularly useful in the development of expert systems. Expert systems are built by capturing the knowledge of humans; however, such knowledge is known to be qualitative and inexact. Experts may be only partially knowledgeable about the problem domain, or data may not be fully available, yet decisions are still expected. In these situations, educated guesses have to be made to provide solutions to problems. This is exactly where fuzzy logic can be employed as a tool to deal with imprecision and qualitative aspects that are associated with problem solving. The rationale for the use of fuzzy logic in expert systems may be summarized as follows:

- The knowledge base of expert systems encapsulates the knowledge of human experts
- Human knowledge is often inexact and qualitative, and fuzzy descriptors (e.g., large, small, fast, poor, fine) are commonly used in its communication

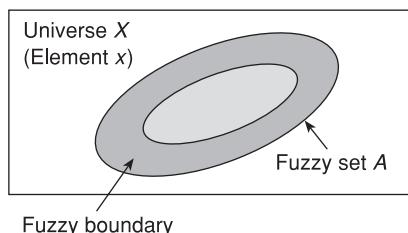
- Problem description of the user may not be exact
- Knowledge base from experts may not be complete, yet satisfactory decisions are still expected
- Educated guesses need to be made in some situations.

## 2.3 Fuzzy sets

The theory of fuzzy logic can be developed using the concepts of fuzzy sets similar to how the theory of crisp bivalent logic can be developed using the concepts of crisp sets. Specifically, there exists an isomorphism between sets and logic. In view of this, a good foundation in fuzzy sets is necessary to develop the theory of fuzzy logic. In this section, the mathematics of fuzzy sets is presented.

A fuzzy set is a set without clear or sharp (crisp) boundaries or without binary membership characteristics. Unlike an ordinary set where each object (or element) either belongs or does not belong to the set, partial membership in a fuzzy set is possible. In other words, there is a “softness” associated with the membership of elements in a fuzzy set. An example of a fuzzy set could be “the set of narrow streets in Vancouver.” There are streets that clearly belong to the above set, and others that cannot be considered as narrow. Since the concept of “narrow” is not precisely defined (for example, < 2m), there will be a “gray” zone in the associated set where the membership is not quite obvious. As another example, consider the variable “temperature”. It can take a fuzzy value (e.g., cold, cool, tepid, warm, hot). A *fuzzy value* such as “warm” is a *fuzzy descriptor*. It may be represented by a fuzzy set because any temperature that is considered to represent “warm” belongs to this set and any other temperature does not belong to the set. Still, one cannot realistically identify a precise temperature interval (e.g., 25°C to 30°C), which is a crisp set, to represent warm temperatures.

Let  $X$  be a set that contains every set of interest in the context of a given class of problems. This is called the *universe of discourse* (or simply universe), whose elements are denoted by  $x$ . A fuzzy set  $A$  in  $X$  may be represented by a Venn diagram as in Figure 2.1. Generally, the elements  $x$  are not numerical quantities. For analytical convenience, however, the elements  $x$  are assigned real numerical values.



**Figure 2.1:** Venn diagram of a fuzzy set

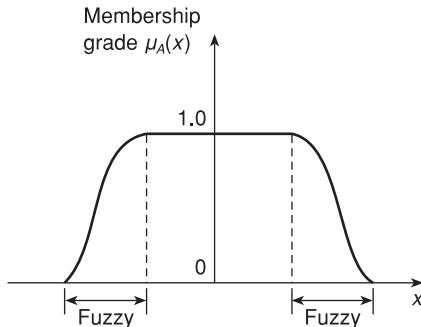


Figure 2.2: The membership function of a fuzzy set

### 2.3.1 Membership function

A fuzzy set may be represented by a membership function. This function gives the grade (degree) of membership within the set, of any element of the universe of discourse. The membership function maps the elements of the universe on to numerical values in the interval  $[0, 1]$ . Specifically,

$$\mu_A(x): X \rightarrow [0, 1] \quad (2.1)$$

where  $\mu_A(x)$  is the membership function of the fuzzy set  $A$  in the universe in  $X$ . Stated in another way, fuzzy set  $A$  is a set of ordered pairs:

$$A = \{(x, \mu_A(x)); x \in X, \mu_A(x) \in [0, 1]\} \quad (2.2)$$

The membership function  $\mu_A(x)$  represents the grade of possibility that an element  $x$  belongs to the set  $A$ . It follows that a membership function is a *possibility function* and not a probability function. A membership function value of zero implies that the corresponding element is definitely not an element of the fuzzy set. A membership function value of unity means that the corresponding element is definitely an element of the fuzzy set. A grade of membership greater than 0 and less than 1 corresponds to a non-crisp (or fuzzy) membership, and the corresponding elements fall on the fuzzy boundary of the set. The closer the  $\mu_A(x)$  is to 1 the more the  $x$  is considered to belong to  $A$ , and similarly the closer it is to 0 the less it is considered to belong to  $A$ . A typical membership function is shown in Figure 2.2.

Note: A crisp set is a special case of the fuzzy set, where the membership function can take the two values 1 (membership) and 0 (non-membership) only. The membership function of a crisp set is given the special name *characteristic function*.

### 2.3.2 Symbolic representation

A universe of discourse, and a membership function which spans the universe, completely define a fuzzy set. We are not usually interested in the elements

with zero grade of membership. The crisp set formed by leaving out all the boundary elements (i.e., leading and trailing elements in the membership function) that have a zero membership grade and retaining all the remaining elements is termed the *support set* of the particular fuzzy set. The support set is a crisp subset of the universe. Furthermore, a fuzzy set is clearly a subset of its support set. A fuzzy set may be symbolically represented as:

$$A = \{x | \mu_A(x)\} \quad (2.3)$$

If the universe is discrete with elements  $x_i$  then a fuzzy set may be specified using a convenient form of notation due to Zadeh, in which each element is paired with its grade of membership in the form of a “formal series” as

$$A = \mu_A(x_1)/x_1 + \mu_A(x_2)/x_2 + \dots + \mu_A(x_i)/x_i + \dots$$

or

$$A = \sum_{x_i \in X} \frac{\mu_A(x_i)}{x_i} \quad (2.4)$$

If the universe is continuous, an equivalent form of notation is given in terms of a “symbolic” integration

$$A = \int_{x \in X} \frac{\mu_A(x)}{x} \quad (2.5)$$

It is important to emphasize that both the series in (2.4) and the integral in (2.5) are symbolic shorthand forms of notation; to highlight this, no differential symbol  $d(\cdot)$  is used in (2.5).

Four examples are now given to illustrate various types of fuzzy sets and their representations.

### Example 2.2

Suppose that the universe of discourse  $X$  is the set of positive integers (or natural numbers). Consider the fuzzy set  $A$  in this discrete universe, given by the Zadeh notation:

$$A = 0.2/3 + 0.3/4 + 1.0/5 + 0.2/6 + 0.1/7$$

This set may be interpreted as a fuzzy representation of the integer 5.

**Example 2.3**

Consider the continuous universe of discourse  $X$  representing the set of real numbers ( $\mathbb{R}$ ). The membership function:

$$\mu_A(x) = 1/[1 + (x - a)^{10}]$$

defines a fuzzy set  $A$  whose elements  $x$  vaguely represent those satisfying the crisp relation  $x = a$ . This fuzzy set corresponds to a *fuzzy relation*.

**Example 2.4**

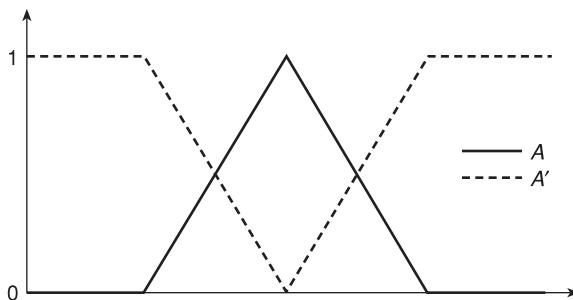
Linguistic terms such as “tall men”, “beautiful women”, and “fast cars are” *fuzzy labels* which can be represented by fuzzy sets. Note that their membership in a set is subjective and not crisp. The fuzzy terms such as “tall”, “beautiful”, and “fast” are fuzzy descriptors. These are also *linguistic variables*, which take fuzzy values.

**Example 2.5**

Consider the linguistic variable “very tall”. This can assume a fuzzy value and can be represented by a fuzzy set (or a membership function). The fuzzy adjective “very” is called a *linguistic hedge*. Other examples of linguistic hedges are slightly, extremely, more or less, slightly, and highly.

## 2.4 Fuzzy logic operations

It is well known that the “complement”, “union”, and “intersection” of crisp sets correspond to the logical operations NOT, OR, and AND, respectively, in the corresponding crisp, bivalent logic. Furthermore, it is known that, in the crisp, bivalent logic, the union of a set with the complement of a second set represents an “implication” of the first set by the second set. Set inclusion (subset) is a special case of implication in which the two sets belong to the same universe. These operations (connectives) may be extended to fuzzy sets for corresponding use in fuzzy logic and fuzzy reasoning. For fuzzy sets, the applicable connectives must be expressed in terms of the membership functions of the sets which are operated on. In view of the isomorphism between fuzzy sets and fuzzy logic, both the set operations and the logical connectives can be addressed together. Some basic operations that can be defined on fuzzy sets and the corresponding connectives of fuzzy logic are described next.



**Figure 2.3:** Fuzzy-set complement of fuzzy logic NOT

Several methods are available to define the intersection and the union of fuzzy sets. The classical ones suggested by Zadeh are widely used, because of their simplicity and the analogy with crisp sets (binary logic).

#### 2.4.1 Complement (negation, NOT)

Consider a fuzzy set  $A$  in a universe  $X$ . Its complement  $A'$  is a fuzzy set whose membership function is given by

$$\mu_{A'}(x) = 1 - \mu_A(x) \quad \text{for all } x \in X \quad (2.6)$$

The complement in fuzzy sets corresponds to the negation (NOT) operation in fuzzy logic, just as in crisp logic, and is denoted by  $\bar{A}$  where  $A$  now is a fuzzy logic proposition (or a fuzzy state).

A graphic (membership function) representation of the complement of a fuzzy set (or negation of a fuzzy state) is given in Figure 2.3.

#### Example 2.6

As an example, consider the fuzzy set of “hot temperatures”. This may be represented by the membership function shown using a solid line in Figure 2.4. This is the set containing all values of hot temperature in a specified universe. In fuzzy logic then, this membership function can represent the fuzzy logic state “hot” or the fuzzy statement, “the temperature is hot”. The complement of the fuzzy set is represented by the dotted line in Figure 2.3. This is the set containing all temperature values that are not hot, in the given universe. As before, this membership function can also represent the fuzzy logic state “not hot” or the fuzzy logic statement “the temperature is not hot”. Furthermore, the solid line can represent a fuzzy temperature value and the dotted line can represent another (complementary) fuzzy temperature value. In this manner, various concepts of fuzzy sets and fuzzy logic can be quantified by the use of membership functions.

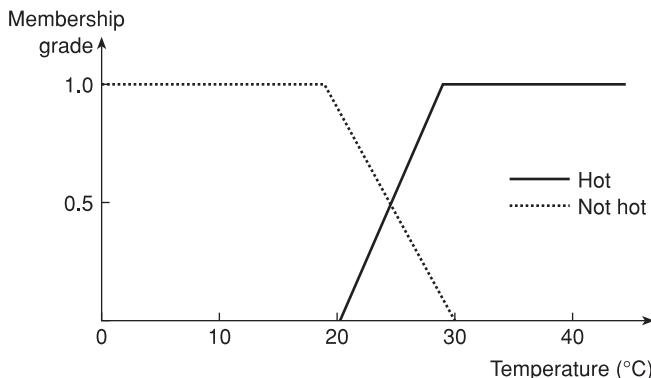


Figure 2.4: An example of fuzzy logic NOT

#### 2.4.2 Union (disjunction, OR)

Consider two fuzzy sets  $A$  and  $B$  in the same universe  $X$ . Their union is a fuzzy set containing all the elements from both sets, in a “fuzzy” sense. This set operation is denoted by  $\cup$ . The membership function of the resulting set  $A \cup B$  is given by

$$\mu_{A \cup B}(x) = \max[(\mu_A(x), \mu_B(x))] \quad \forall x \in X \quad (2.7)$$

The union corresponds to a logical OR operation (called *Disjunction*), and is denoted by  $A \vee B$ , where  $A$  and  $B$  are fuzzy states or fuzzy propositions. The rationale for the use of *max* to represent a fuzzy-set union is that, because element  $x$  may belong to one set or the other, the larger of the two membership grades should govern the outcome (union). Furthermore, this is consistent with the union of crisp sets, when carried out using characteristic functions (Note:  $\max(1, 0) = 1$ ). Similarly, the appropriateness of using *max* to represent fuzzy logic operation “OR” should be clear. Specifically, since either of the two fuzzy states (or propositions) would be applicable, the larger of the corresponding two membership grades should be used to represent the outcome. A graphic (membership function) representation of the union of two fuzzy sets (or the logical combination OR of two fuzzy states in the same universe) is given in Figure 2.5.

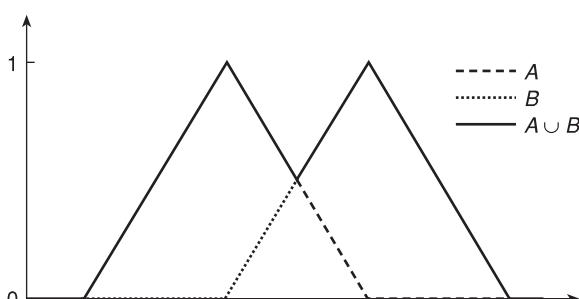


Figure 2.5: Fuzzy-set union or fuzzy logic OR

### Example 2.7

Consider a universe representing the driving speeds on a highway, in km/h. Suppose that the fuzzy logic state “Fast” is given by the discrete membership function

$$F = 0.6/80 + 0.8/90 + 1.0/100 + 1.0/110 + 1.0/120$$

and the fuzzy state “Medium” is given by

$$M = 0.6/50 + 0.8/60 + 1.0/70 + 1.0/80 + 0.8/90 + 0.4/100$$

Then the combined fuzzy condition “Fast OR Medium” is given by the membership function

$$F \vee M = 0.6/50 + 0.8/60 + 1.0/70 + 1.0/80 + 0.8/90 + 1.0/100 + 1.0/110 + 1.0/120$$

### Example 2.8

Consider a universe representing room temperature (in °C) and another universe representing relative humidity (%). Suppose that an acceptable temperature is given by the membership function

$$T = 0.4/16 + 0.8/18 + 1.0/20 + 1.0/22 + 0.8/24 + 0.5/26$$

and an acceptable humidity is given by the membership function

$$H = 0.2/0 + 0.8/20 + 1.0/40 + 0.6/60 + 0.2/80$$

Then the fuzzy condition “Acceptable Temperature OR Acceptable Humidity” is given by the following membership function with a two-dimensional universe:

		Temperature (°C)					
		16	18	20	22	24	26
Relative humidity (%)	0	0.4	0.8	1.0	1.0	0.8	0.5
	20	0.8	0.8	1.0	1.0	0.8	0.8
	40	1.0	1.0	1.0	1.0	1.0	1.0
	60	0.6	0.8	1.0	1.0	0.8	0.6
	80	0.4	0.8	1.0	1.0	0.8	0.8

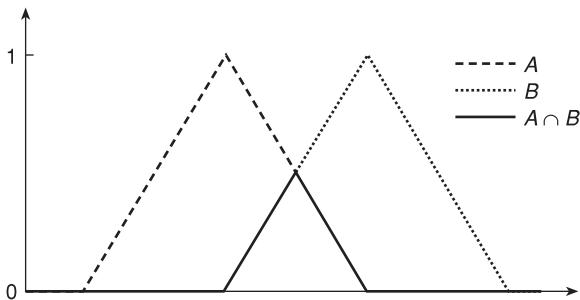


Figure 2.6: Fuzzy-set intersection or fuzzy logic AND

### 2.4.3 Intersection (conjunction, AND)

Again, consider two fuzzy sets  $A$  and  $B$  in the same universe  $X$ . Their intersection is a fuzzy set containing all the elements that are common to both sets, in a “fuzzy” sense. This set operation is denoted by  $\cap$ . The membership function of the resulting set  $A \cap B$  is given by

$$\mu_{A \cap B}(x) = \min[(\mu_A(x), \mu_B(x))] \quad \forall x \in X \quad (2.8)$$

The union corresponds to a logical AND operation (called *Conjunction*), and is denoted by  $A \wedge B$ , where  $A$  and  $B$  are fuzzy states or fuzzy propositions. The rationale for the use of *min* to represent fuzzy-set intersection is that, because the element  $x$  must simultaneously belong to both sets, the smaller of the two membership grades should govern the outcome (intersection).

Furthermore, this is consistent with the intersection of crisp sets, when carried out using characteristic functions (Note:  $\min(1, 0) = 0$ ). Similarly, the appropriateness of using *min* to represent fuzzy logic operation “AND” should be clear. Specifically, since both fuzzy states (or propositions) should be simultaneously present, the smaller of the corresponding two membership grades should be used to represent the outcome. A graphic (membership function) representation of the intersection of two fuzzy sets (or the logical combination AND of two fuzzy states in the same universe) is given in Figure 2.6.

#### Example 2.9

The membership function representing the required speed of operation of a robotic manipulator is shown in Figure 2.7(a). The membership function of the required power is shown in Figure 2.7(b). Then the membership function representing the required speed and the required power is shown in Figure 2.7(c).

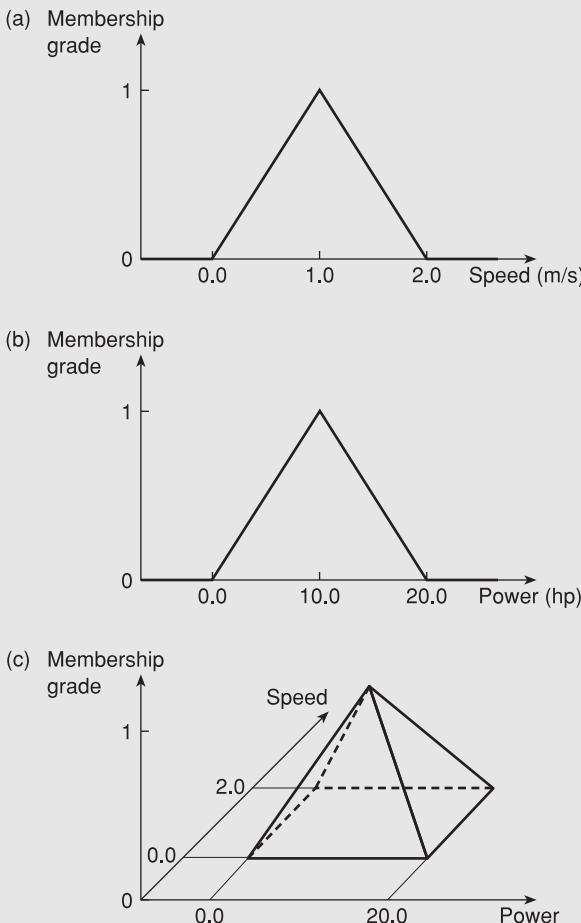


Figure 2.7: (a) Required speed; (b) Required power; (c) Required speed and power

#### 2.4.4 Basic laws of fuzzy logic

Laws of logic govern the equivalence of various types of logical expressions consisting of the basic connectives AND, OR, and NOT. Fuzzy sets and the basic fuzzy operations as we have defined previously, satisfy some general laws. On one hand, these laws may be interpreted as properties of fuzzy sets. On the other hand, they may be interpreted as properties of fuzzy operations. As in the case of crisp sets and bivalent logic, these laws apply to both fuzzy sets and fuzzy logic. They may be used to simplify fuzzy logic expressions and, in particular, to process a fuzzy logic knowledge base.

Consider three general fuzzy sets  $A$ ,  $B$ , and  $C$ , defined in a common universe  $X$ . Let  $\phi$  denote the null set (a set with no elements, and hence having a membership function of zero value). With this notation, some important

**Table 2.1: Some properties of fuzzy sets**

Property name	Relation
Commutativity	$A \cap B = B \cap A$ $A \cup B = B \cup A$
Associativity	$(A \cap B) \cap C = A \cap (B \cap C)$ $(A \cup B) \cup C = A \cup (B \cup C)$
Distributivity	$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
Absorption	$A \cup (A \cap B) = A$ $A \cap (A \cup B) = A$
Idempotency (Idem = same; potent = power) (Similar to unity or identity operation)	$A \cup A = A$ $A \cap A = A$
Exclusion: Law of excluded middle Law of contradiction	$A \cup A' \subset X$ $A \cap A' \supset \emptyset$
DeMorgan's Laws	$(A \cap B)' = A' \cup B'$ $(A \cup B)' = A' \cap B'$
Boundary conditions	$A \cup X = X$ $A \cap X = A$ $A \cup \emptyset = A$ $A \cap \emptyset = \emptyset$

properties of fuzzy sets are summarized in Table 2.1. These properties are either obvious or may be easily verified, using the definitions of complement ('), union ( $\cup$ ), and intersection ( $\cap$ ) that were given previously. DeMorgan's Laws are particularly useful in simplifying (processing) expressions of sets (and logic). It is noted that the *laws of exclusion*, which are satisfied by crisp sets, are not satisfied by fuzzy sets. In particular, the union of a fuzzy set  $A$  and its complement (i.e.,  $A \cup A'$ ) is not equal to the universal set, but is a subset of the universal set (i.e.,  $\subset X$ ). Furthermore, the intersection of a fuzzy set  $A$  and its complement (i.e.,  $A \cap A'$ ) is not a null set and may have non-zero membership values; so,  $A \cap A' \supset \emptyset$ . These exclusion properties of a fuzzy set are not a shortcoming. In fact they enable fuzzy logic to be more practical and versatile than crisp logic, as we shall see.

### Example 2.10

To illustrate how the law of excluded middle is violated by fuzzy sets, consider the example shown in Figure 2.8. Here a triangular membership function (broken line) is used to represent a fuzzy set  $A$ . The complement  $A'$  is shown by a dotted line, which is obtained by subtracting the original membership function from 1. Next, the union of  $A$  and  $A'$  is performed using the *max* operation.

The resulting membership function is given by the solid line. Note that the result is not uniformly equal to 1 (i.e., not equal to  $\mu_x$ ). In particular, the middle part of  $\mu_x$  is excluded in the result.

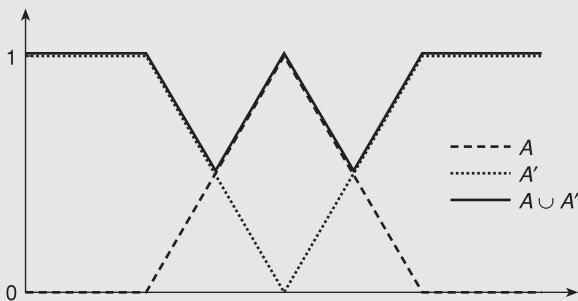


Figure 2.8: An example of excluded middle in fuzzy sets

### Example 2.11

Suppose that  $A$  denotes the set of “my true statements”. Then, in the universe of “all my statements”, the complement of  $A$  (i.e.,  $A'$ ) denotes the set of “my false statements”.

- Using  $A$ ,  $A'$ , and the operations of union ( $\cup$ ) and equality ( $=$ ) of sets, write an equation (in sets) that is equivalent to the logic proposition: “All my statements are false.”
- Show that according to bivalent, crisp logic (i.e., assuming that  $A$  is a crisp set) the statement in (a) above is a contradiction.

Suppose that fuzzy logic (using fuzzy sets) is used with the statement in (a), where the complement operation is given by  $\mu_{A'} = 1 - \mu_A$  and the union operation is represented by “max”. For what membership values of  $A$  does the statement in (a) hold?

#### **Solution**

- $A \cup A' = A'$
- If the statement in (a) is true, then  $A \neq \emptyset$ . But for crisp logic, the law of excluded middle:  $A \cup A' = X$  is satisfied. Hence, from (a)  $A' = X$  and hence  $A = \emptyset$ . This is a contradiction, as  $A \neq \emptyset$ .
- According to (a), with fuzzy logic, we have  $\max(\mu_A, 1 - \mu_A) = 1 - \mu_A$  which holds if and only if  $\mu_A \leq 0.5$ .

Note that this means “my true statements” are at best completely fuzzy and generally tend to be false. This is closer to the reality, as there is nothing wrong in the statement of (a), from a subjective humanistic viewpoint. This is clearly an advantage of fuzzy logic.

## 2.5 Generalized fuzzy operations

The three fuzzy operations described above are special cases, which are commonly used due to their simplicity and effectiveness. More general versions of these operations are available. These are primarily of analytical significance.

### 2.5.1 Generalized fuzzy complement

A generalized complement operation, denoted by  $C$ , should satisfy the following axioms:

- (1) Boundary conditions:  $C(\phi) = X$  and  $C(X) = \phi$  where  $X$  is the universal set (having membership function  $\mu_X = 1$ ) and  $\phi$  is the null set (having membership function  $\mu_\phi = 0$ ).
- (2) Non-increasing: For two fuzzy sets  $A$  and  $B$  with membership functions  $a = \mu_A(x)$  and  $b = \mu_B(x)$  in the same universe  $X$ , if  $a > b$ , then  $C(a) \leq C(b)$ . Another way of expressing this property is by using the concept of subsets ( $\subset$ ). Specifically, if  $A \supset B$  then  $C(A) \subset C(B)$ .
- (3) Involutive:  $C(C(A)) = A$ . Note that this *property of involution* is analogous to *double negation*.

Clearly, the special case of complement  $(1 - \mu)$ , which was given previously, satisfies these three axioms. Note that the complement operation is applied to a membership function (which takes values in the interval  $[0, 1]$ ) and it generates another membership function (which also takes values in the interval  $[0, 1]$ ). Symbolically, this fact is given by

$$C: [0, 1] \rightarrow [0, 1]$$

A fuzzy complement operation, due to Sugeno, which satisfies the requirements of generalized fuzzy complement as given above, is

$$C(a) = \frac{1 - a}{1 + pa} \quad (2.9)$$

where  $a$  is a membership function of a fuzzy set and  $p$  is a real parameter which can take any value from  $-1$  to  $\infty$ ; i.e.,  $p \in (-1, \infty)$ .

Another fuzzy complement operation, due to Yager, which satisfies the requirements of generalized fuzzy complement, is

$$C(a) = (1 - a^p)^{1/p} \quad (2.10)$$

where  $a$  is a membership function of a fuzzy set and  $p$  is a real parameter which can take any value from  $0$  to  $\infty$ ; i.e.,  $p \in (0, \infty)$ .

### Example 2.12

Axiom 1 of complement gives the boundary condition:  $C(\phi) = X$ . Hence,  $C(C(\phi)) = C(X)$ .

Axiom 3 (involution property) gives:  $C(C(\phi)) = \phi$ .

It follows that:  $C(X) = \phi$ , which is the second boundary condition. Hence, one of the boundary conditions need not be explicitly specified, in view of the involution property.

## 2.5.2 Triangular norms

Membership functions take values in the interval  $[0, 1]$ . A *binary operation* (of fuzzy sets and fuzzy logic) is one where the operation is applied to “two” membership functions (or membership grades) at a time, resulting in one membership function (or grade). The associated mapping is indicated by:

$$f: [0, 1] \times [0, 1] \rightarrow [0, 1] \quad (2.11)$$

This fact should be clear because each one of the two membership functions that are operated upon (i.e., the two operands or arguments) takes values in the *real-unit interval*  $[0, 1]$  and the membership function that results from this operation also takes values in the unit interval. The operation itself is denoted by “ $f$ ” in equation (2.11). The two binary operations which we have encountered thus far are *min* and *max*. In view of the “triangular” nature of the mapping given by equation (2.11) the operation is termed a *triangular norm*. These ideas can be generalized as outlined below. Note that the binary operations carried out on membership functions are *fuzzy aggregation* operations. They aggregate two (or more) membership functions into one.

### 2.5.2.1 T-norm (*generalized intersection*)

As listed in Table 2.2, a triangular norm (or T-norm) is a binary operation or function of the type given by equation (2.11). Consider two membership functions that are given by  $a = \mu_A(x)$  and  $b = \mu_B(x)$ . Then, the T-norm operation may be represented by  $T(a, b)$  or more commonly  $aTb$ . The T-norm possesses the following four properties:

- (1) It is nondecreasing in each argument.
- (2) It satisfies commutativity.
- (3) It satisfies associativity.
- (4) It satisfies the boundary condition  $aT1 = a$ , where  $a$  denotes a general membership function.

It can be shown that a second boundary condition

$$aT0 = 0$$

**Table 2.2: Some properties of a triangular norm**

Item description	T-norm (triangular norm)	S-norm (T-conorm)
Function	$T: [0, 1] \times [0, 1] \rightarrow [0, 1]$	Same
Nondecreasing in each argument	If $b \geq a, d \geq c$ then $bTd \geq aTc$	Same
Commutative	$aTb = bTa$	Same
Associative	$(aTb)Tc = aT(bTc)$	Same
Boundary conditions	$aT1 = a$ $aT0 = 0$ with $a, b, c, d \in [0, 1]$	$aS0 = a$ $aS1 = 1$
Examples	Conventional: $\min(a, b)$ Product: $ab$ Bounded max (bold intersection): $\max[0, a + b - 1]$ General: $1 - \min[1, ((1 - a)^p + (1 - b)^p)^{1/p}]$ $p \geq 1$ $\max[0, (\lambda + 1)(a + b - 1) - \lambda ab]$ $\lambda \geq -1$	Conventional: $\max(a, b)$ Set addition: $a + b - ab$ Bounded min (bold union): $\min[1, a + b]$ General: $\min(1, (a^p + b^p)^{1/p})$ $p \geq 1$ $\min[1, a + b + \lambda ab]$ $\lambda \geq -1$
DeMorgan's Laws	$aSb = 1 - (1 - a) T(1 - b)$ $aTb = 1 - (1 - a) S(1 - b)$	

is also satisfied. To prove this, set  $a = 0$  in Property 4. Hence,  $0T1 = 0$ . However, from Property 2, we have  $1T0 = 0$ . Now, in view of Property 1, it follows that  $1T0 \geq xT0$ . Since  $xT0$  is in the real interval  $[0, 1]$  it cannot be less than 0. Hence,  $xT0 = 0$ .

Three examples of the T-norm are given in Table 2.2. Note that the *min* operation is a special case of T-norm. Hence we may interpret the T-norm as a *generalized intersection* that may be applied to fuzzy sets (or membership functions).

### 2.5.2.2 S-norm or triangular conorm (generalized union)

The complementary operation of the T-norm is called the S-norm. As indicated in Table 2.2, this is also a triangular norm and a binary operation of the form given by equation (2.11). The S-norm operation may be represented by  $S(a, b)$  or more commonly  $aSb$ . The S-norm possesses the following four properties:

- (1) It is nondecreasing in each argument.
- (2) It satisfies commutativity.
- (3) It satisfies associativity.
- (4) It satisfies the boundary condition,  $aS0 = a$ , where  $a$  denotes a general membership function.

In parallel with the T-norm, it can be shown that a second boundary condition,  $aS1 = 1$ , is also automatically satisfied.

**Exercise:** Using the four properties of S-norm, show that the boundary condition  $aS1 = 1$  holds.

Three examples of S-norm are given in Table 2.2. Note that the *max* operation is a special case of S-norm and that this norm may be interpreted as a *generalized union* for fuzzy sets (and fuzzy logic). The theory of fuzzy logic may be generalized by using the S-norm and the T-norm rather than *max* and *min* operations. DeMorgan's Laws, as given in Table 2.2, are also satisfied by these two generalized norms.

### Example 2.13

Using DeMorgan's Laws, determine the S-norm corresponding to the T-norm:  $\min(x, y)$ .

#### Solution

Use DeMorgan's Law:  $xSy = 1 - (1 - x)T(1 - y)$

Direct substitution of *min* for *T* gives:

$$\begin{aligned} xSy &= 1 - \min[(1 - x), (1 - y)] = 1 - (1 - y) \quad \text{if } x \geq y \\ &\quad = 1 - (1 - y) \quad \text{if } x < y \end{aligned}$$

Hence

$$\begin{aligned} xSy &= x \quad \text{if } x \geq y \\ &= y \quad \text{if } x < y \end{aligned}$$

Hence

$$xSy = \max(x, y)$$

### Example 2.14

Prove that the “minimum” operator is the largest T-norm and the “maximum” operator is the smallest S-norm. Specifically, show that

$$\min(x, y) \geq xTy$$

and

$$\max(x, y) \leq xSy$$

From this result prove that

$$xTy \leq \min(x, y) \leq \max(x, y) \leq xSy$$

**Solution**

From the nondecreasing property and boundary condition of a T-norm we have

$$\begin{aligned} xTy &\leq 1Ty = y \\ xTy &\leq xT1 = x \end{aligned}$$

Hence

$$\begin{aligned} xTy &\leq y \\ xTy &\leq x \end{aligned}$$

It follows that  $xTy$  is less than or equal to both  $x$  and  $y$ . In other words

$$xTy \leq \min(x, y)$$

From the nondecreasing property and boundary condition of an S-norm we have

$$\begin{aligned} xSy &\geq 0Sy = y \\ xSy &\geq xS0 = x \end{aligned}$$

Hence

$$\begin{aligned} xSy &\geq y \\ xSy &\geq x \end{aligned}$$

It follows that

$$xSy \geq \max(x, y)$$

But

$$\max(x, y) \geq \min(x, y)$$

Hence

$$xSy \geq \max(x, y) \geq \min(x, y) \geq xTy \quad (2.12)$$

### 2.5.3 Set inclusion ( $A \subset B$ )

The concept of subset (or a set “included” in another set) in crisp sets may be conveniently extended to the case of fuzzy sets. Specifically, a fuzzy set  $A$  is considered a subset of another fuzzy set  $B$ , in universe  $X$ , if and only if

$$\mu_A(x) \leq \mu_B(x) \quad \text{for all } x \in X \quad (2.13)$$

This is denoted by  $A \subset B$ .

### 2.5.3.1 Grade of inclusion

In the context of fuzzy logic, a set may be partially included in another set. Then it is convenient to define a grade of inclusion of a fuzzy set  $A$  in another fuzzy set  $B$ . This may be interpreted as the membership function of the *fuzzy relation*  $A \subset B$ . Specifically, we use

$$\begin{aligned} \mu_{A \subset B}(x) &= 1 && \text{if } \mu_A(x) \leq \mu_B(x) \\ &= \mu_A(x)T\mu_B(x) && \text{otherwise} \end{aligned} \quad (2.14)$$

Equation (2.14) gives a general definition for grade of inclusion, and accordingly we have used T-norm. In a specific application, we may use the most conservative T-norm, which is *min*, or a stronger (more local) T-norm such as the *product*. Usually *min* is preferred in more general (or global) and weaker relations (weaker interactions) of fuzzy sets, and *product* is preferred in more specific (or local) and stronger relations (stronger interactions) of fuzzy sets. It may be verified that equation (2.14) is equivalent to

$$\mu_{A \subset B}(x) = \sup[c \in [0, 1] \mid \mu_A(x)Tc \leq \mu_B(x)] \quad (2.15)$$

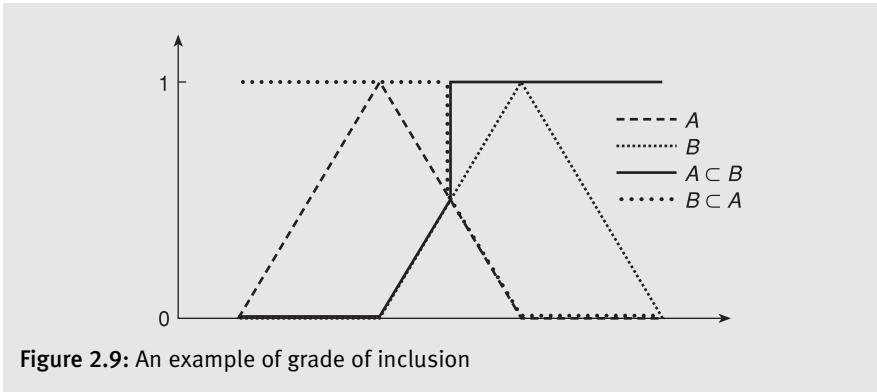
Here the *supremum* operation is denoted by *sup*. This obtains the maximum value of a function over a continuous (or piecewise continuous) interval of its variable. In equation (2.15), the function ( $c$ ) is the variable itself. In determining the *sup*, the variable  $c$  is varied from 0 to 1 continuously, and the maximum value of  $c$  in this interval that satisfies the condition  $\mu_A(x)Tc \leq \mu_B(x)$  is the answer.

#### Example 2.15

When *min* is used as the T-norm, equation (2.14) becomes

$$\begin{aligned} \mu_{A \subset B}(x) &= 1 && \text{if } \mu_A(x) \leq \mu_B(x)* \\ &= \mu_B(x) && \text{otherwise} \end{aligned} \quad (2.14)^*$$

Consider two fuzzy sets  $A$  and  $B$ , shown by the broken line and the dotted line, in Figure 2.9. According to equation (2.14)\*, the grade of inclusion of  $A$  in  $B$  is given by the solid line and the grade of inclusion of  $B$  in  $A$  is given by the crossed line.



### 2.5.4 Set equality ( $A = B$ )

The equality of two fuzzy sets is a special case of set inclusion. A fuzzy set  $A$  is equal to another fuzzy set  $B$ , in universe  $X$ , if and only if

$$\mu_A(x) = \mu_B(x) \quad \text{for all } x \in X \quad (2.16)$$

This is denoted by  $A = B$ . If  $A \subset B$  and  $A \neq B$ , then  $A$  is called a *proper subset* of  $B$ .

#### 2.5.4.1 Grade of equality

A grade of equality for two fuzzy sets may be defined similar to the grade of inclusion. This may be interpreted as the membership function of the *fuzzy relation*  $A = B$ . Specifically, we use

$$\begin{aligned} \mu_{A=B}(x) &= 1 && \text{if } \mu_A(x) = \mu_B(x) \\ &= \mu_A(x)T\mu_B(x) && \text{otherwise} \end{aligned} \quad (2.17)$$

As before, in a specific application, we may use either min or product to represent the T-norm in this general definition.

## 2.6 Implication (if-then)

An if-then statement (a rule) is called an “implication”. In a knowledge-based system, the knowledge base is commonly represented using if-then rules. In particular, a knowledge base in fuzzy logic may be expressed by a set of linguistic rules of the if-then type, containing fuzzy terms. In fact a fuzzy rule is a *fuzzy relation*. A knowledge base containing several fuzzy rules is also a relation, which is formed by combining (aggregating) the individual rules according to how they are interconnected.

Consider a fuzzy set  $A$  defined in a universe  $X$  and a second fuzzy set  $B$  defined in another universe  $Y$ . The fuzzy implication “If  $A$  then  $B$ ” is denoted

by  $A \rightarrow B$ . Note that in this fuzzy rule,  $A$  represents some “fuzzy” situation, and is the *condition* or the *antecedent* of the rule. Similarly,  $B$  represents another fuzzy situation, and is the *action* or the *consequent* of the fuzzy rule. The fuzzy rule  $A \rightarrow B$  is a fuzzy relation. Since the elements of  $A$  are defined in  $X$  and the elements of  $B$  are defined in  $Y$ , the elements of  $A \rightarrow B$  are defined in the *Cartesian product space*  $X \times Y$ . This is a two-dimensional space represented by two orthogonal axes ( $x$ -axis and  $y$ -axis), and gives the domain in which the fuzzy rule (or fuzzy relation) is defined. Since  $A$  and  $B$  can be represented by membership functions, an additional orthogonal axis is needed to represent the membership grade.

Fuzzy implication may be defined (interpreted) in several ways. Two definitions of fuzzy implication are:

*Method 1:*

$$\mu_{A \rightarrow B}(x, y) = \min[(\mu_A(x), \mu_B(y))] \quad (2.18)$$

$$\forall x \in X, \forall y \in Y$$

*Method 2:*

$$\mu_{A \rightarrow B}(x, y) = \min[1, \{1 - \mu_A(x) + \mu_B(y)\}] \quad (2.19)$$

$$\forall x \in X, \forall y \in Y$$

These two methods are approaches for obtaining the membership function of the particular fuzzy relation given by an *if-then* rule (implication). Note that the first method gives an expression that is symmetric with respect to  $A$  and  $B$ . This is not intuitively satisfying because “implication” is not a commutative operation (specifically,  $A \rightarrow B$  does not necessarily satisfy  $B \rightarrow A$ ). In practice, however, this method provides a good, robust result. The second method has an intuitive appeal because in crisp bivalent logic,  $A \rightarrow B$  has the same truth table as  $[(\text{NOT } A) \text{ OR } B]$  and hence they are equivalent. Note that in equation (2.19), the membership function is upper-bounded to 1 using the *bounded sum* operation, as required (a membership grade cannot be greater than 1). The first method is more commonly used because it is simpler to use and often provides quite accurate results. Fuzzy implication is further examined next.

### 2.6.1 Considerations of fuzzy implication

We have noticed that in classical propositional calculus, we can show by writing truth tables that the implication  $A \rightarrow B$  is equivalent to “(Not  $A$ ) Or  $B$ ,” which may be written as

$$A \rightarrow B \equiv \bar{A} \vee B. \quad (2.20)$$

In fact, this is also equivalent to

$$A \rightarrow B \equiv (A \wedge B) \vee \bar{A} \quad (2.21)$$

because

$$(A \wedge B) \vee \bar{A} = (A \vee \bar{A}) \wedge (B \vee \bar{A}) = X \wedge (B \vee \bar{A}) = B \vee \bar{A} = \bar{A} \vee B.$$

Note that we used the distributivity of  $\vee$  over  $\wedge$  and the commutativity of  $\vee$ ; where  $-$ ,  $\wedge$ , and  $\vee$  represent (classical) logic operations “not,” “and,” and “or,” respectively. The fuzzy implication can be viewed as a generalization of the crisp operation where  $A$  and  $B$  represent fuzzy propositions, and the operations  $-$ ,  $\wedge$ , and  $\vee$  represent fuzzy complement, fuzzy intersection, and fuzzy union, respectively.

In a local sense,  $A \rightarrow B$  can also be taken to mean  $\bar{A} \rightarrow \bar{B}$ . In this local implication,  $A \rightarrow B$  would also mean  $B \rightarrow A$  (because  $A \rightarrow B$  is identical to  $\bar{B} \rightarrow \bar{A}$ ; and hence  $\bar{A} \rightarrow \bar{B}$  is identical to  $B \rightarrow A$ ). For crisp binary logic, the truth tables show that  $A \rightarrow B$  is not identical to  $B \rightarrow A$ .

In a local sense then, we may take the implication to mean:

$$A \rightarrow B \equiv A \wedge B \quad (2.22)$$

The interpretation (2.22) of implication is symmetric, and satisfies  $A \rightarrow B$  and  $\bar{A} \rightarrow \bar{B}$ . This interpretation can be thought of as a stronger form of (2.21) in view of the fact that (2.21) represents “(A And B) Or (Not A)” and (2.22) represents “A And B.” Specifically, (2.21) is equal to “(2.22) OR something.” Also, the former interpretation means “A entails B” and the latter means that “A is coupled with B.” Note that in the relations (2.20)–(2.22) the sets  $A$  and  $B$  represent fuzzy logic propositions, and are defined in different universes  $X$  and  $Y$ .

The interpretation (2.22) is in fact what is given by equation (2.18), which is due to Mamdani. Furthermore, the interpretation (2.20) is what is given by equation (2.19), which is due to Lukasiewicz. These and several other methods of fuzzy implication, as derived from relations (2.20)–(2.22), are given below.

(a) Larsen implication

$$\mu_{A \rightarrow B}(x, y) = \mu_A(x)\mu_B(y) \\ \forall x \in X, \forall y \in Y$$

(b) Mamdani implication

$$\mu_{A \rightarrow B}(x, y) = \min[\mu_A(x), \mu_B(y)] \\ \forall x \in X, \forall y \in Y$$

(c) Zadeh implication

$$\mu_{A \rightarrow B}(x, y) = \max[\min\{\mu_A(x), \mu_B(y)\}, 1 - \mu_A(x)] \\ \forall x \in X, \forall y \in Y$$

(d) Dienes–Rescher implication

$$\mu_{A \rightarrow B}(x, y) = \max[1 - \mu_A(x), \mu_B(y)] \\ \forall x \in X, \forall y \in Y$$

(e) Lukasiewicz implication (Bounded sum)

$$\mu_{A \rightarrow B}(x, y) = \min[1, 1 - \mu_A(x) + \mu_B(y)] \\ \forall x \in X, \forall y \in Y$$

**Table 2.3: Some interpretations of fuzzy implication**

Representation of $A \rightarrow B$	Meaning	Comments	Examples
$\bar{A} \vee B$	(Not $A$ ) Or $B$	<ul style="list-style-type: none"> <li>■ Same as in crisp binary logic</li> <li>■ A weaker (global) implication, meaning “<math>A</math> entails <math>B</math>”</li> </ul>	<ul style="list-style-type: none"> <li>■ Dienes–Rescher implication <math>\max\{1 - \mu_A(x), \mu_B(y)\}</math></li> <li>■ Lukasiewicz implication <math>\min\{1, 1 - \mu_A(x) + \mu_B(y)\}</math></li> </ul>
$(A \wedge B) \vee \bar{A}$	( $A$ And $B$ ) Or (Not $A$ )	Same as above	<ul style="list-style-type: none"> <li>■ Zadeh implication <math>\max[\min\{\mu_A(x), \mu_B(y)\}, 1 - \mu_A(x)]</math></li> </ul>
$A \wedge B$	$A$ And $B$	<ul style="list-style-type: none"> <li>■ A stronger (local) implication, meaning “<math>A</math> is coupled with <math>B</math>”</li> </ul>	<ul style="list-style-type: none"> <li>■ Mamdani implication <math>\min\{\mu_A(x), \mu_B(y)\}</math></li> <li>■ Larsen implication <math>\mu_A(x)\mu_B(y)</math></li> </ul>

where  $\mu_{A \rightarrow B}(x, y)$  is the membership function of the relation, which represents fuzzy implication  $A \rightarrow B$ . These interpretations are summarized in Table 2.3.

It is possible to establish a relationship of ordering among these five fuzzy implication operations. We proceed as below.

Since  $0 \leq 1 - \mu_A(x) \leq 1$  and  $0 \leq \mu_B(y) \leq 1$ , we have

$$\max[1 - \mu_A(x), \mu_B(y)] \leq 1 - \mu_A(x) + \mu_B(y) \text{ and } \max[1 - \mu_A(x), \mu_B(y)] \leq 1.$$

Hence

$$\max[1 - \mu_A(x), \mu_B(y)] \leq \min[1, 1 - \mu_A(x) + \mu_B(y)].$$

Moreover, since  $\min[\mu_A(x), \mu_B(y)] \leq \mu_B(y)$ , we have

$$\max[\min\{\mu_A(x), \mu_B(y)\}, 1 - \mu_A(x)] \leq \max[\mu_B(y), 1 - \mu_A(x)].$$

Also, since  $\max(p, a) \geq p$ , we have

$$\max[\min\{\mu_A(x), \mu_B(y)\}, 1 - \mu_A(x)] \geq \min[\mu_A(x), \mu_B(y)].$$

Furthermore, since “min” is the largest T-norm and the “product” is another T-norm, we have

$$\min[\mu_A(x), \mu_B(y)] \geq \mu_A(x)\mu_B(y).$$

From these relationships it is noted that

$$\text{Larsen} \subseteq \text{Mamdani} \subseteq \text{Zadeh} \subseteq \text{Dienes–Rescher} \subseteq \text{Lukasiewicz}.$$

It follows that Larsen implication is the most local (strongest) one among them. Mamdani implication is the most widely used one in fuzzy systems applications, and particularly in fuzzy control because it is simpler to use,

robust, and generally provides good results. A similar comment can be made about Larsen implication. Unlike the remaining implication operations, Mamdani and Larsen implications are based on the concept that fuzzy IF-THEN rules tend to be local. For example, when we say “IF the target is very near THEN apply the brakes quickly,” we deal with a local situation. It is not generally valid, and for example, cannot be used to evaluate the condition “the target is somewhat near.” Furthermore, according to Mamdani and Larsen implications, “IF the target is very near THEN apply the brakes quickly” also implies “IF the target is not very near THEN don’t apply the brakes quickly.” Since this may be acceptable in a local sense but is not generally valid, a particular type of implication may not be uniformly effective in all types of fuzzy knowledge-based applications.

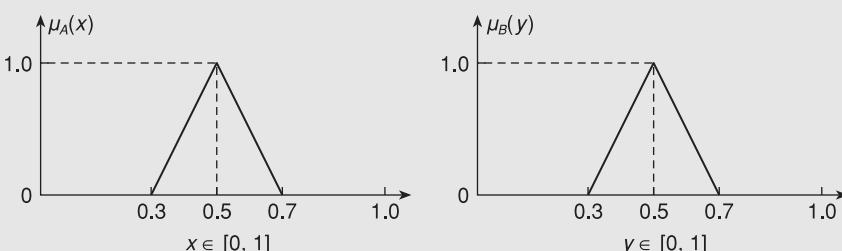
### Example 2.16

Consider the membership functions of fuzzy sets  $A$  and  $B$  as shown in Figure 2.10, and expressed below:

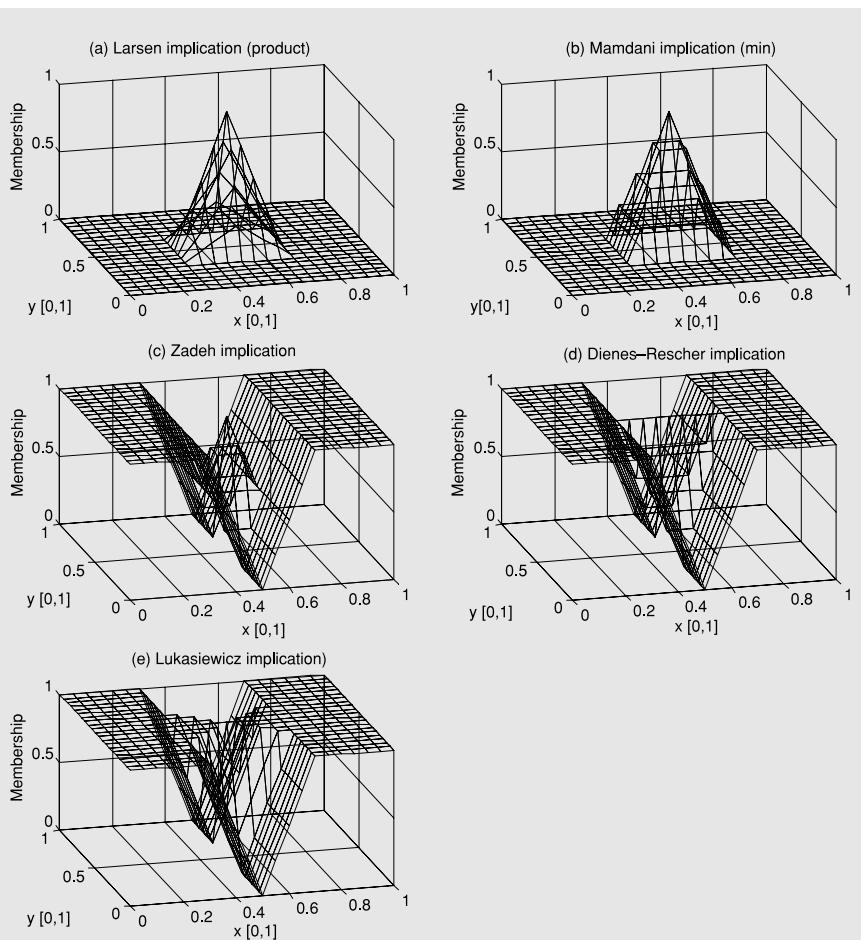
$$\begin{aligned}\mu_A(x) &= \max \left\{ 0, \frac{10x - 3}{2} \right\} \quad 0.3 \leq x \leq 0.5 \\ &= \max \left\{ 0, \frac{7 - 10x}{2} \right\} \quad 0.5 < x \leq 0.7 \\ &= 0 \quad \text{otherwise}\end{aligned}$$

$$\begin{aligned}\mu_B(y) &= \max \left\{ 0, \frac{10y - 3}{2} \right\} \quad 0.3 \leq y \leq 0.5 \\ &= \max \left\{ 0, \frac{7 - 10y}{2} \right\} \quad 0.5 < y \leq 0.7 \\ &= 0 \quad \text{otherwise}\end{aligned}$$

The resulting expressions for the combined membership functions, which represent the five implication relations, are given in (a)–(e) below, and sketched in Figure 2.11.



**Figure 2.10:** Membership functions of fuzzy sets  $A$  and  $B$



**Figure 2.11:** A graphical representation of various fuzzy implications operations

(a) Larsen implication (product or dot operation)

$$\mu_{A \rightarrow B}(x, y) = \begin{cases} \frac{(10x - 3)(10y - 3)}{4} & \text{if } 0.3 \leq x \leq 0.5 \text{ and } 0.3 \leq y \leq 0.5 \\ \frac{(10x - 3)(7 - 10y)}{4} & \text{if } 0.3 \leq x \leq 0.5 \text{ and } 0.5 < y \leq 0.7 \\ \frac{(7 - 10x)(10y - 3)}{4} & \text{if } 0.5 < x \leq 0.7 \text{ and } 0.3 \leq y \leq 0.5 \\ \frac{(7 - 10x)(7 - 10y)}{4} & \text{if } 0.5 < x \leq 0.7 \text{ and } 0.5 < y \leq 0.7 \\ 0 & \text{otherwise} \end{cases}$$

(b) Mamdani implication (min operation)

$$\mu_{A \rightarrow B}(x, y) = \begin{cases} \min\left[\frac{10x - 3}{2}, \frac{10y - 3}{2}\right] & \text{if } 0.3 \leq x \leq 0.5 \text{ and } 0.3 \leq y \leq 0.5 \\ \min\left[\frac{10x - 3}{2}, \frac{7 - 10y}{2}\right] & \text{if } 0.3 \leq x \leq 0.5 \text{ and } 0.5 < y \leq 0.7 \\ \min\left[\frac{7 - 10x}{2}, \frac{10y - 3}{2}\right] & \text{if } 0.5 < x \leq 0.7 \text{ and } 0.3 \leq y \leq 0.5 \\ \min\left[\frac{7 - 10x}{2}, \frac{7 - 10y}{2}\right] & \text{if } 0.5 < x \leq 0.7 \text{ and } 0.5 < y \leq 0.7 \\ 0 & \text{otherwise} \end{cases}$$

(c) Zadeh implication

$$\mu_{A \rightarrow B}(x, y) = \begin{cases} \max\left[\min\left\{\frac{10x - 3}{2}, \frac{10y - 3}{2}\right\}, \frac{5 - 10x}{2}\right] & \text{if } 0.3 \leq x \leq 0.5 \text{ and } 0.3 \leq y \leq 0.5 \\ \max\left[\min\left\{\frac{10x - 3}{2}, \frac{7 - 10y}{2}\right\}, \frac{5 - 10x}{2}\right] & \text{if } 0.3 \leq x \leq 0.5 \text{ and } 0.5 < y \leq 0.7 \\ \frac{5 - 10x}{2} & \text{if } 0.3 \leq x \leq 0.5 \text{ and } (0 \leq y < 0.3 \text{ or } 0.7 < y \leq 1) \\ \max\left[\min\left\{\frac{7 - 10x}{2}, \frac{10y - 3}{2}\right\}, \frac{10x - 3}{2}\right] & \text{if } 0.5 < x \leq 0.7 \text{ and } 0.3 \leq y \leq 0.5 \\ \max\left[\min\left\{\frac{7 - 10x}{2}, \frac{7 - 10y}{2}\right\}, \frac{10x - 3}{2}\right] & \text{if } 0.5 < x \leq 0.7 \text{ and } 0.5 < y \leq 0.7 \\ \frac{10x - 5}{2} & \text{if } 0.5 < x \leq 0.7 \text{ and } (0 \leq y < 0.3 \text{ or } 0.7 < y \leq 1) \\ 1 & \text{otherwise} \end{cases}$$

(d) Dienes–Rescher implication

$$\mu_{A \rightarrow B}(x, y) = \begin{cases} \max\left[\frac{5 - 10x}{2}, \frac{10y - 3}{2}\right] & \text{if } 0.3 \leq x \leq 0.5 \text{ and } 0.3 \leq y \leq 0.5 \\ \max\left[\frac{5 - 10x}{2}, \frac{7 - 10y}{2}\right] & \text{if } 0.3 \leq x \leq 0.5 \text{ and } 0.5 < y \leq 0.7 \\ \frac{5 - 10x}{2} & \text{if } 0.3 \leq x < 0.5 \text{ and } (0 \leq y < 0.3 \text{ or } 0.7 < y \leq 1) \\ \max\left[\frac{10x - 5}{2}, \frac{10y - 3}{2}\right] & \text{if } 0.5 < x \leq 0.7 \text{ and } 0.3 \leq y \leq 0.5 \\ \max\left[\frac{10x - 5}{2}, \frac{7 - 10y}{2}\right] & \text{if } 0.5 < x \leq 0.7 \text{ and } 0.5 < y \leq 0.7 \\ \frac{10x - 5}{2} & \text{if } 0.5 < x \leq 0.7 \text{ and } (0 \leq y < 0.3 \text{ and } 0.7 < y \leq 1) \\ 1 & \text{otherwise} \end{cases}$$

(e) Lukasiewicz implication

$$\mu_{A \rightarrow B}(x, y) = \begin{cases} \min[1, 1 - 5x + 5y] & \text{if } 0.3 \leq x \leq 0.5 \text{ and } 0.3 \leq y \leq 0.5 \\ \min[1, 6 - 5x - 5y] & \text{if } 0.3 \leq x \leq 0.5 \text{ and } 0.5 < y \leq 0.7 \\ \min\left[1, \frac{5 - 10x}{2}\right] & \text{if } 0.3 \leq x < 0.5 \text{ and} \\ & (0 \leq y < 0.3 \text{ or } 0.7 < y \leq 1) \\ \min[1, 5x + 5y - 4] & \text{if } 0.5 < x \leq 0.7 \text{ and } 0.3 \leq y \leq 0.5 \\ \min[1, 1 + 5x - 5y] & \text{if } 0.5 < x \leq 0.7 \text{ and } 0.5 < y \leq 0.7 \\ \min\left[1, \frac{10x - 5}{2}\right] & \text{if } 0.5 < x \leq 0.7 \text{ and} \\ & (0 \leq y < 0.3 \text{ or } 0.7 < y \leq 1) \\ \min\left[1, \frac{10y + 1}{2}\right] & \text{if } (0 \leq x < 0.3 \text{ or } 0.7 < x \leq 1) \\ & \text{and } 0.3 \leq y \leq 0.5 \\ \min\left[1, \frac{9 - 10y}{2}\right] & \text{if } (0 \leq x < 0.3 \text{ or } 0.7 < x \leq 1) \\ & \text{and } 0.5 < y \leq 0.7 \\ 1 & \text{otherwise} \end{cases}$$

The results illustrate the fact that Larsen implication is more local (represented by a narrower region) than the rest, even though it is similar in form to Mamdani implication. Furthermore, the results show that the implications of Zadeh, Dienes–Rescher, and Lukasiewicz are similar in form, Zadeh implication being the most local of the three (i.e., narrowest region of implication).

## 2.7 Some definitions

This section presents some further definitions and concepts of fuzzy sets that are useful in the analysis and application of fuzzy logic.

### 2.7.1 Height of a fuzzy set

The height of a fuzzy set is the maximum value of its membership function. Specifically, for a fuzzy set  $A$  in the universe  $X$ , with membership function  $\mu$ , the height is given by

$$\text{hgt}(A) = \sup_{x \in X} \mu_A(x) \quad (2.23)$$

The height may be termed the *modal grade* of the fuzzy set, and the corresponding element value (i.e., the value of  $x$  where the membership grade peaks) may be termed the *modal element value*, or simply *modal point*.

### 2.7.2 Support set

The support set of a fuzzy set is a crisp set containing all the elements (in the universe) whose membership grade is greater than 0. Specifically, the support set  $S$  of a fuzzy set  $A$  with membership function  $\mu_A(x)$  is given by

$$S = \{x \in X \mid \mu_A(x) > 0\} \quad (2.24)$$

Note that  $S$  is a crisp set containing all the elements  $x$  in the universe  $X$  that satisfy  $\mu_A(x) > 0$ .

### 2.7.3 $\alpha$ -cut of a fuzzy set

The  $\alpha$ -cut of a fuzzy set  $A$  is the crisp set denoted by  $A_\alpha$  formed by those elements of  $A$  whose membership function grade is greater than or equal to a specified threshold value  $\alpha$ . Clearly, it is necessary that  $\alpha$  has to be limited to a value in the interval 0 to 1. Specifically,

$$A_\alpha = \{x \in X \mid \mu_A(x) \geq \alpha\}, \quad \alpha \in [0, 1] \quad (2.25)$$

in which  $X$  is the universe to which  $A$  belongs.

A strong  $\alpha$ -cut can be defined by replacing  $\geq$  with  $>$ . Specifically,

$$A_\alpha = \{x \in X \mid \mu_A(x) > \alpha\}, \quad \alpha \in [0, 1] \quad (2.25)^*$$

When  $\alpha = 0$ , the strong  $\alpha$ -cut becomes the support set of the fuzzy set, as given by equation (2.24).

### 2.7.4 Representation theorem

It should be intuitively clear that a fuzzy set  $A$  might be recomposed or “represented” by means of all  $\alpha$ -cuts of the fuzzy set. The representation theorem expresses this fact, and is given by

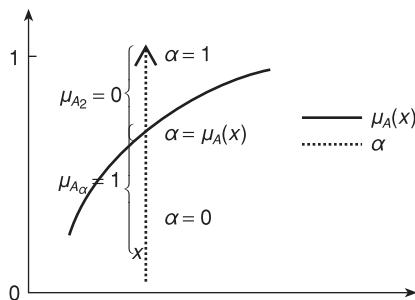
$$\mu_A(x) = \sup_{\alpha \in [0,1]} [\alpha \mu_{A_\alpha}(x)] \quad (2.26)$$

where “sup” denotes the supremum operation, which in the present case gives the overall maximum of the specified function as  $\alpha$  is varied continuously from 0 to 1.

**Proof:** Suppose that at  $x = x_1$ ,  $\mu_A(x) = \alpha_1$ . Then, for  $\alpha < \alpha_1$  we have  $\mu_{A_\alpha}(x_1) = 1$  and for  $\alpha > \alpha_1$  we have  $\mu_{A_\alpha}(x_1) = 0$ . It follows that the supremum value of the RHS of equation (2.13) is equal to  $\alpha_1$ , which is equal to  $\mu_A(x_1)$ . QED.

The proof of the representation theorem is graphically shown in Figure 2.12.

Note that the representation theorem provides an analytical link between a fuzzy set and crisp sets ( $\alpha$ -cuts), and is an analytical means of reconstructing a fuzzy set using crisp sets. This does not mean, however, that a fuzzy set is equivalent to a sequence of crisp sets (because the value of  $\alpha$  itself is a membership grade in this interpretation). The representation theorem is also known as a *resolution identity* because a fuzzy set is “resolved” into a set of  $\alpha$ -cuts (an infinite number).



**Figure 2.12:** Graphical proof of the representation theorem

## 2.8 Fuzziness and fuzzy resolution

The resolution of a fuzzy variable refers to the number of fuzzy states which it can assume. This is given by the number of modes (peaks) in the membership function of the particular variable. Fuzziness quantifies the degree of fuzziness of a set. This measure is particularly useful in hierarchical (multi-level) fuzzy systems. In particular, a high level of fuzziness may be tolerated at high levels of a hierarchical system, while even a slight fuzziness at the lowest level can lead to inaccurate actions. For a multi-modal membership function, the fuzziness has to be quantified on a per-mode basis. Accordingly, there is some connection between fuzzy resolution and fuzziness. In this section we will discuss the meanings of “fuzzy resolution” and “degree of fuzziness.”

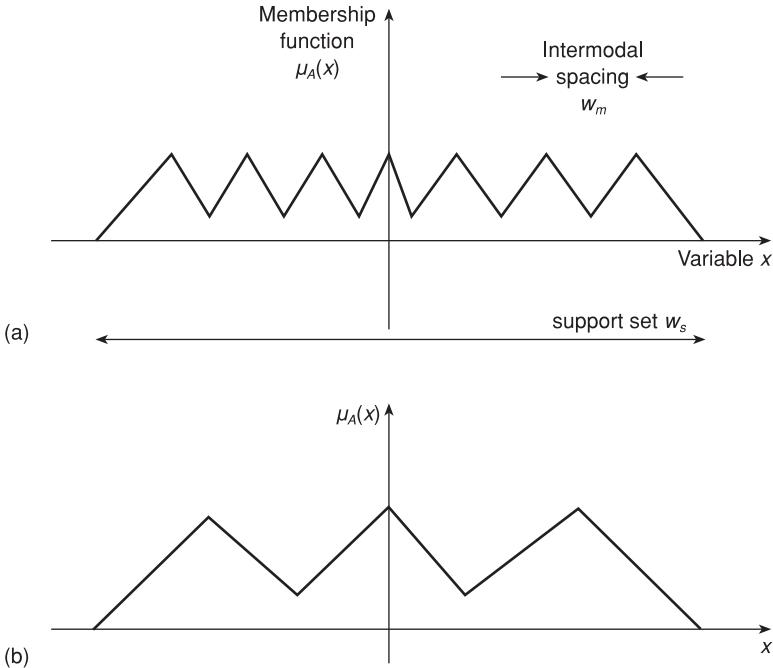
### 2.8.1 Fuzzy resolution

Consider the variable “temperature.” It may assume a number of fuzzy states like cold, cool, tepid, warm, and hot. Each state may be represented by a membership function. The total number of possible fuzzy states of a variable, within a given support set (or knowledge base), represents the fuzzy resolution of the variable. For instance, suppose that the knowledge base of a fuzzy decision-making system is expressed by the set of linguistic rules:

$$\text{Else } [ \text{If } E_1^i \text{ Then If } E_2^j \text{ Then If } \dots E_n^k \text{ Then } C_1^a \text{ And } C_q^b \text{ And } \dots C_s^c ] \quad (2.27)$$

where the condition variable (antecedent)  $e_j$  may assume  $m$  discrete fuzzy states  $E_j^1, E_j^2, \dots, E_j^m$  for  $j = 1, 2, \dots, n$  and similarly each action variable (consequent)  $c_j$  may assume  $r$  discrete fuzzy states  $C_j^1, C_j^2, \dots, C_j^r$  for  $j = 1, 2, \dots, p$ . Here, the integer value  $m$  represents the fuzzy resolution of  $E_j$  and similarly  $r$  represents the fuzzy resolution of  $C_j$ .

The fuzzy resolution of a variable may be interpreted through the nature of the membership function of that variable as well. Specifically, each fuzzy state of the variable will have a corresponding modal point in the membership



**Figure 2.13:** An interpretation of fuzzy resolution. (a) High fuzzy resolution.  
 (b) Low fuzzy resolution

function (typically, having a peak membership grade). Then, the number of such modal points in the membership function (within its support set) corresponds to the number of fuzzy states of the variable (within the knowledge base). Accordingly the *intermodal spacing* of the membership function may be used as a measure of the fuzzy resolution of the variable. This interpretation is illustrated in Figure 2.13. Then an appropriate quantitative definition for fuzzy resolution  $r_f$  would be

$$r_f = \frac{w_s}{w_m} \quad (2.28)$$

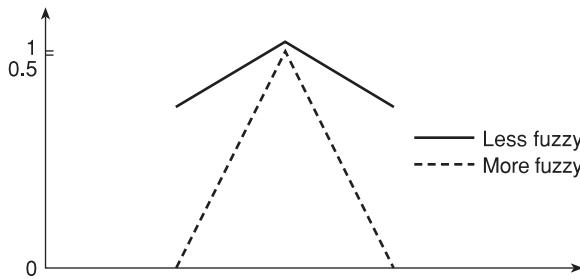
in which

$w_s$  = width of the support set of  $A$

$w_m$  = intermodal spacing

The membership function of  $A$  is given by  $\mu_A(x): \mathfrak{R} \rightarrow [0, 1]$  as usual, where the universe  $X$  of elements  $x$  is the real line  $\mathfrak{R}$ , and it is mapped onto the unit interval  $[0, 1]$  by the membership function.

Note that as the fuzzy resolution increases (i.e., becomes finer), the information resolution also increases, and the corresponding knowledge base becomes more appropriate for a lower hierarchical level. When the member-



**Figure 2.14:** Illustration of fuzziness

ship function of a fuzzy state is trapezoidal (which is usually the case for the two end states of a fuzzy variable) and not triangular, it is difficult to establish a single modal point. In such situations, the centroidal location may be considered as the representative modal point of the particular fuzzy state, in the present definition of fuzzy resolution.

## 2.8.2 Degree of fuzziness

The degree of fuzziness of a fuzzy set is a measure of the difficulty of ascertaining whether each element in the support set belongs within or outside the fuzzy set. In other words, it is a measure of the difficulty of ascertaining the element membership. If the membership grade of an element is less than 0.5, the possibility of the element being outside the fuzzy set is greater, and hence it is easier to exclude that element from the set. Similarly, if the membership grade of an element is greater than 0.5, the possibility of the element being inside the fuzzy set is greater, and hence it is easier to include that element into the set. Clearly then, the most fuzzy elements are those whose membership grade is 0.5. According to this idea, Figure 2.14 shows two fuzzy sets, one of which (membership function shown by the solid line) is fuzzier than the other (membership function shown by the broken line). Many quantitative measures are available for the degree of fuzziness of a set. Three of them are defined next.

### 2.8.2.1 Measures of fuzziness

If the membership grade of an element is close to unity, the element is almost definitely a member of the set. Conversely, if the membership grade of an element is close to zero, the element is nearly outside the set. From this observation it is clear that the membership of an element  $x$  in a set  $A$  is most fuzzy when the membership grade  $\mu_A(x) = 0.5$ . Accordingly, the fuzziness of a set may be measured by the closeness of its elements to the membership grade of 0.5. In view of this observation, an appropriate measure of fuzziness of a fuzzy set  $A$  would be

$$A_1 = \int_{x \in S} f(x) dx \quad (2.29)$$

in which the nonnegative function  $f(x)$  is defined in the support set  $S$  of the fuzzy set  $A$  by

$$f(x): S \rightarrow [0, 0.5]$$

with

$$\begin{aligned} f(x) &= \mu_A(x) && \text{for } \mu_A(x) \leq 0.5 \\ &= 1 - \mu_A(x) && \text{otherwise} \end{aligned} \quad (2.30)$$

The value of  $A_1$  is larger when the membership function is closer to the grade 0.5. As required then, this measure gives the closeness of the membership function to the most fuzzy grade (0.5).

Another measure of fuzziness may be given as follows:

$$A_2 = \int_{x \in S} |\mu_A(x) - \mu_{A1/2}(x)| dx \quad (2.31)$$

Here  $\mu_{A1/2}$  is the  $1/2$ -cut of  $A$ , and hence, this measure gives the distance (separation) of the membership function from its  $1/2$ -cut. Since,

$$\begin{aligned} \mu_{1/2} &= 0 && \text{for values of } x \text{ where } \mu_A(x) < 0.5 \\ \mu_{1/2} &= 1 && \text{for values of } x \text{ where } \mu_A(x) > 0.5 \end{aligned}$$

it should be clear that equation (2.29) is identical to (2.31), and hence

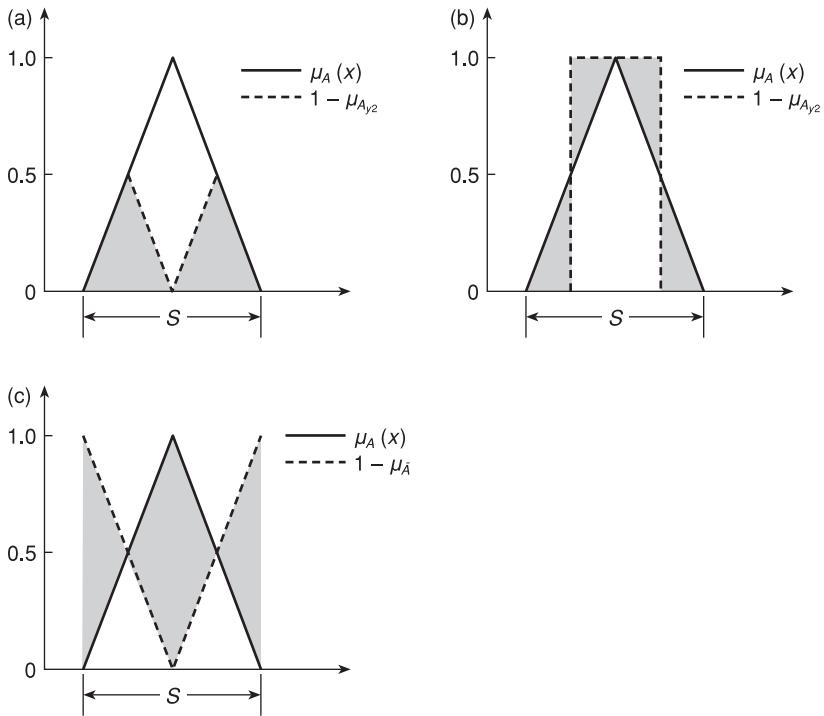
$$A_1 = A_2$$

This fact is illustrated in the example of Figure 2.15.

Since the fuzziness of a set is measured by the closeness of its elements to the membership grade of 0.5, it may be measured by the “inverse” of the distance to the grade of 0.5. Specifically, we define

$$\begin{aligned} A_3 &= 2 \int_{x \in S} |\mu_A(x) - 0.5| dx \\ &= \int_{x \in S} |2\mu_A(x) - 1| dx \\ &= \int_{x \in S} |\mu_A(x) - \bar{\mu}_A(x)| dx \end{aligned} \quad (2.32)$$

Note that  $\bar{\mu}_A = 1 - \mu_A$ . The area  $A_3$  is given as the shaded area in the example of Figure 2.15(c). In fact this area is twice the complement of area  $A_1$ . It follows that the three measures of fuzziness as given here are identical. Specifically,



**Figure 2.15:** Illustration of three measures of fuzziness: (a) Closeness to grade 0.5. (b) Distance from  $^{1/2}$ -cut. (c) Inverse of distance from the complement

$$A_1 = A_2 = \frac{1}{2}(S - A_3) \quad (2.33)$$

where  $S$  is the length of the support set, as shown in Figure 2.15.

These measures of fuzziness assume continuous (piecewise) membership functions, but they may be extended to discrete membership functions as well. Also, since the present definitions of fuzziness are given on a “per mode” basis, the expressions should be divided by the number of modal points in  $A$  to account for the multi-modal case. It should be clear from Figure 2.14 that, for a given support set length, the higher the fuzzy resolution the lower the degree of fuzziness, in general.

### Example 2.17

Consider a fuzzy set  $A$  in the universe  $\mathfrak{R}$ , the real line, whose membership function is given by

$$\begin{aligned} \mu_A(x) &= 1 - 2|x - 1| & \text{for } |x - 1| \leq 0.5 \\ &= 0 & \text{otherwise} \end{aligned}$$

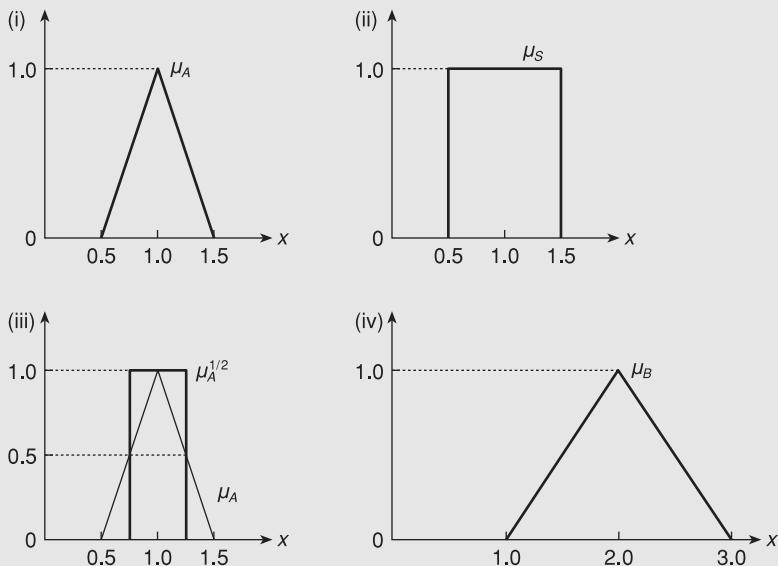
- Sketch this membership function.
- What is the support set of  $A$ ?
- What is the  $1/2$ -cut of  $A$ ?
- Another fuzzy set,  $B$ , is given by the membership function

$$\begin{aligned}\mu_B(x) &= 1 - |x - 2| \quad \text{for } |x - 2| \leq 1 \\ &= 0 \quad \text{otherwise}\end{aligned}$$

Sketch this membership function.

Which one of the sets  $A$  and  $B$  is fuzzier? Explain your answer.

### **Solution**



$$A_{1/2} = \{x | \mu_A(x) \geq 0.5\} = [0.75, 1.25]$$

**Figure 2.16:** (i)  $\mu_A$ , (ii) Support set of  $A$ , (iii)  $A_{1/2}$ , (iv)  $\mu_B$

If we apply the measure of fuzziness given in equation (2.31) to both  $A$  and  $B$  as shown in Figure 2.17, we have:

$$\text{Fuzziness of } A = 4 \times 0.5 \times 0.25 \times 0.5 = 0.25$$

$$\text{Fuzziness of } B = 4 \times 0.5 \times 0.5 \times 0.5 = 0.5.$$

Hence, fuzziness of  $B = 2 \times$  fuzziness of  $A$ .

This concludes that  $B$  is fuzzier than  $A$ , as is clear from Figure 2.17.

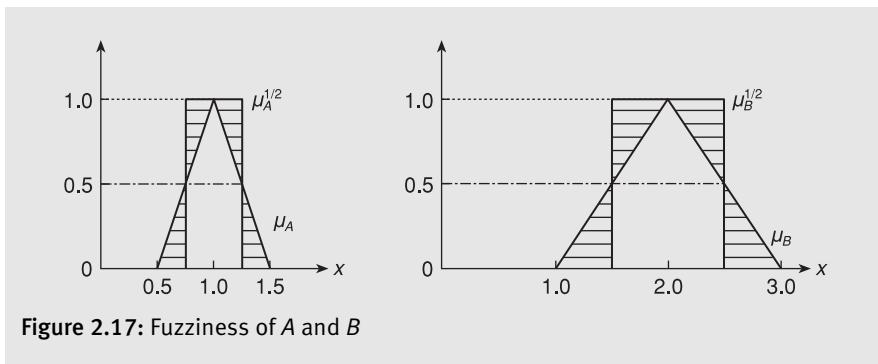


Figure 2.17: Fuzziness of  $A$  and  $B$

## 2.9 Fuzzy relations

First consider an example in crisp sets and binary logic. Specifically, the Boolean relation  $a = x \cdot \bar{y} + z$  is equivalent to the logical expression (proposition) “ $x$  AND NOT  $y$  OR  $z$ ” and is denoted by  $a$  (Note: “and” is “.” and “or” is “+”). An equivalent expression may be written using three sets and the operations Complement, Intersection, and Union, but we will not do so here. A truth table to represent the given Boolean (or logical) expression may be formed, as given in Table 2.4.

Now note the rows where the last column of the truth table has values 1. These rows correspond to the combinations of  $x, y, z$  for which the given expression  $a$  is “True.” This fact may be expressed in the Boolean form as:

$$a = \bar{x} \cdot \bar{y} \cdot z + \bar{x} \cdot y \cdot z + x \cdot \bar{y} \cdot \bar{z} + x \cdot \bar{y} \cdot z + x \cdot y \cdot z$$

Accordingly, we have proved that

$$a = x \cdot \bar{y} + z = \bar{x} \cdot \bar{y} \cdot z + \bar{x} \cdot y \cdot z + x \cdot \bar{y} \cdot \bar{z} + x \cdot \bar{y} \cdot z + x \cdot y \cdot z$$

Table 2.4: Truth table for  $a = x \cdot \bar{y} + z$

$x$	$y$	$z$	$\bar{y}$	$x \times \bar{y}$	$a$
0	0	0	1	0	0
0	0	1	1	0	1
0	1	0	0	0	0
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	0	0	0
1	1	1	0	0	1

The truth table of a logical expression gives the various circumstances (states) of the individual variables of the expression under which the given expression is “True.” In particular, a logical expression (a proposition) such as  $x \cdot \bar{y} + z$  may be expressed as an “OR” combination of all the constituents that are “True” (i.e., equal to 1) in its truth table. When a constituent is “True” then the given expression is “True.” In summary, when the result column of the truth table shown in Table 2.4 has the value 1, this means that the logical (Boolean) relation  $x \cdot \bar{y} + z = 1$  is satisfied (True, or valid).

In Boolean logic (and crisp sets) there are only two truth values (True, False; 1, 0; Member, Non-member). A relation in Boolean, crisp logic is valid when its truth value is 1 (i.e., True). Fuzzy logic (and fuzzy sets) may be interpreted as a generalization of Boolean logic, with infinite possibilities of “truth values” ranging from 0 to 1. Accordingly, a fuzzy logic (or fuzzy set) relation is viewed as a generalization of a Boolean crisp-logic relation, and may be represented by a membership function. The membership grade at a particular point of the membership function gives the degree of validity (or truth value in a general sense) that this point is on the fuzzy relation. In particular, the concept of fuzzy relation may be applied to expressions (operations) such as NOT  $A$ ,  $A$  AND  $B$ ,  $A$  OR  $B$ ,  $A = B$ ,  $A \rightarrow B$ , and so on, which we have discussed.

Concepts of crisp logic and Boolean algebra, as exemplified in Table 2.4, are applied in the design of logic circuitry and digital hardware. Similarly, the extensions of these concepts to fuzzy logic are applicable in fuzzy logic hardware.

### Example 2.18

Consider a fuzzy logic proposition  $A$ , with membership function  $\mu_A$ . Then  $\mu_A$  may be taken to represent the degree of validity of the relation “ $A$  is True.” Similarly, the complement membership function  $\mu_{\bar{A}} = 1 - \mu_A$  represents the degree of validity of the fuzzy relation “ $\bar{A}$  is True” or equivalently “ $A$  is Not True.”

### Example 2.19

Consider a fuzzy logic proposition “ $A$  AND  $B$ .” If  $\mu_A(x)$  is the membership function of  $A$ , and  $\mu_B(y)$  is the membership function of  $B$ , then using min to represent AND, the membership function of  $A$  AND  $B$  is given by

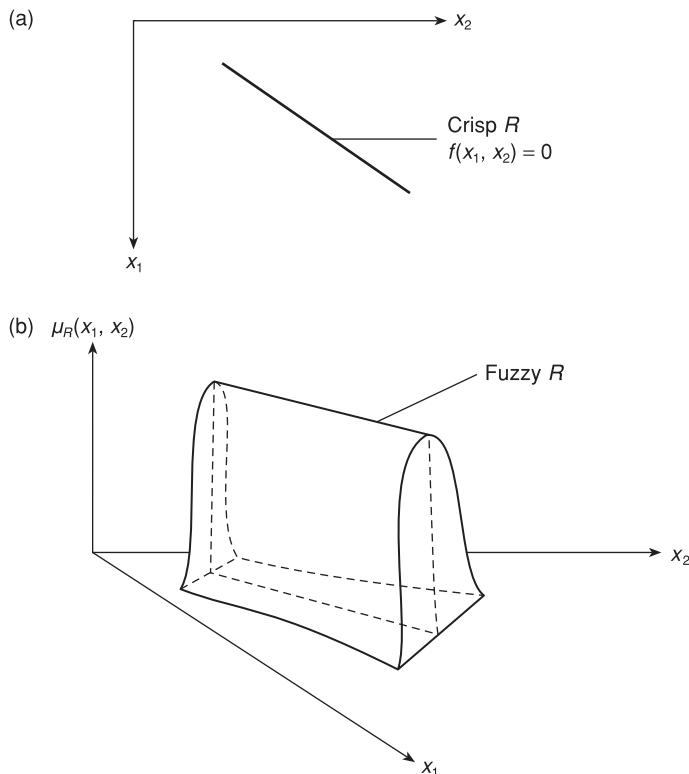
$$R(x, y) = \min[\mu_A(x), \mu_B(y)]$$

This membership function represents the degree of validity of the relation “ $A$  AND  $B$  is True.”

### 2.9.1 Analytical representation of a fuzzy relation

From the foregoing discussion we understand that any membership function represents a fuzzy relation in the universe (or domain, or space) of definition of the particular membership function. The space can be one-dimensional; say, the real line  $\mathfrak{R}$  as in the case of  $\mu_R(x)$ , and this gives a 1-D fuzzy relation. The idea can be extended to higher dimensions.

Consider two universes  $X_1 = \{x_1\}$  and  $X_2 = \{x_2\}$ . A crisp set consisting of a subset of ordered pairs  $(x_1, x_2)$  is a crisp relation  $R$  in the two-dimensional Cartesian product space  $X_1 \times X_2$ . We may imagine that a truth value of 1 is attached to each of these ordered pairs, giving the characteristic function (special case of membership function) of the crisp relation. An example for the case of continuous variables is shown in Figure 2.18(a). Analogously, a fuzzy set  $R$  consisting of a subset of ordered pairs  $(x_1, x_2)$  is a fuzzy relation in the Cartesian product space  $X_1 \times X_2$ . The relation  $R$  is represented by its membership function  $\mu_R(x_1, x_2)$ . An example of a fuzzy relation (continuous case) is shown in Figure 2.18(b). This concept can be extended in a straightforward manner to fuzzy relations in the  $n$ -dimensional Cartesian space  $X_1 \times X_2 \times \dots \times X_n$ .



**Figure 2.18:** Relation  $R$  in a two-dimensional space (plane): (a) A crisp relation.  
(b) A fuzzy relation

### Example 2.20

Consider the fuzzy set  $R$  in the universe  $X_1 \times X_2$  given by the membership function:

$$\mu_R(x_1, x_2) = 1/[1 + 100(x_1 - 3x_2)^4]$$

This is a fuzzy relation approximately representing the crisp relation  $x_1 = 3x_2$ . Particularly note that all elements satisfying  $x_1 = 3x_2$  have a unity grade of membership and hence they are definitely in the set  $R$ . Elements satisfying, for example,  $x_1 = 3.1x_2$  have membership grades less than 1; their membership in  $R$  is non-crisp, soft, or approximate. The further away the elements are from the straight line  $x_1 = 3x_2$ , the softer (less crisp, less possible) the membership of those elements in  $R$ .

#### 2.9.2 Cartesian product of fuzzy sets

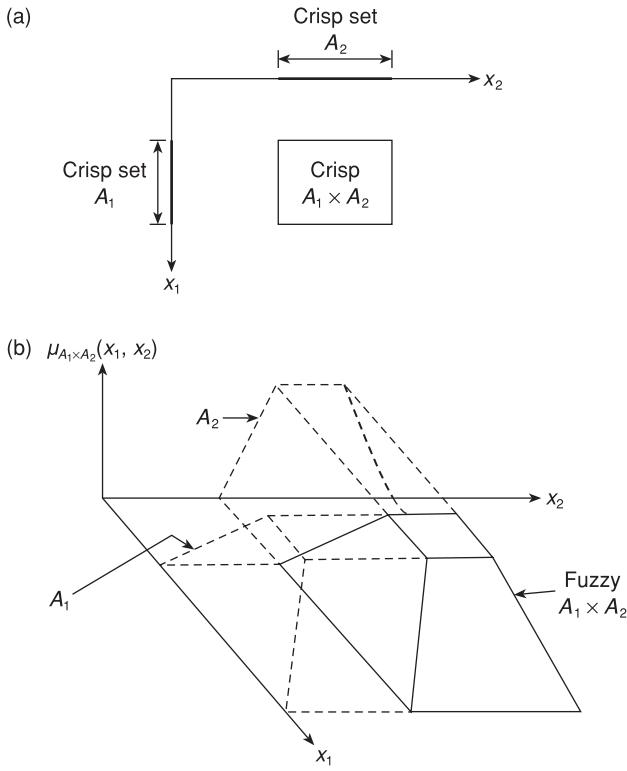
Consider a crisp set  $A_1$  defined in the universe  $X_1$  and a second crisp set  $A_2$  defined in a different (independent, orthogonal) universe  $X_2$ . The Cartesian product  $A_1 \times A_2$  is the rectangular area shown in Figure 2.19(a). This is a subset of the Cartesian product space  $X_1 \times X_2$  defined in the usual manner, which is the entire 2-D space (plane) containing the two axes  $x_1$  and  $x_2$ .

Next consider a fuzzy set  $A_1$  in the universe  $X_1$  and a second fuzzy set  $A_2$  in a different universe  $X_2$ . The Cartesian product  $A_1 \times A_2$  is then a fuzzy subset of the Cartesian space  $X_1 \times X_2$ . Its membership function is given by

$$\mu_{A_1 \times A_2}(x_1, x_2) = \min[\mu_{A_1}(x_1), \mu_{A_2}(x_2)] \quad \forall x_1 \in X_1, \forall x_2 \in X_2 \quad (2.34)$$

This is the *cross product* of  $A$  “and”  $B$ , considered together at a given point in the  $X \times Y$  space. Note that the *min* combination applies here because each element  $(x_1, x_2)$  in the Cartesian product is formed by taking both elements  $x_1$  “and”  $x_2$  together (an “AND” operation), not just one or the other. Clearly, when the possibility of two independent states occurring together is considered, the lower of the possibility values (membership grades) of the two individual states should apply. Cartesian product  $A \times B$  provides the region of definition of the two fuzzy sets  $A$  and  $B$ . It is a subset of the Cartesian product of their support sets, which in turn is a subset of the Cartesian product of their universes ( $X \times Y$ ). An example of a Cartesian product of two fuzzy sets in the continuous case is shown in Figure 2.19(b).

The Cartesian product of two fuzzy sets is a fuzzy relation, which is an AND operation dealing with two different universes. Accordingly, it is also identical to the method of *fuzzy implication* given by equation (2.18). Note that a fuzzy relation can be quite general and coupled within the universes



**Figure 2.19:** Cartesian product ( $A_1 \times A_2$ ) relation of: (a) Two crisp sets.  
(b) Two fuzzy sets

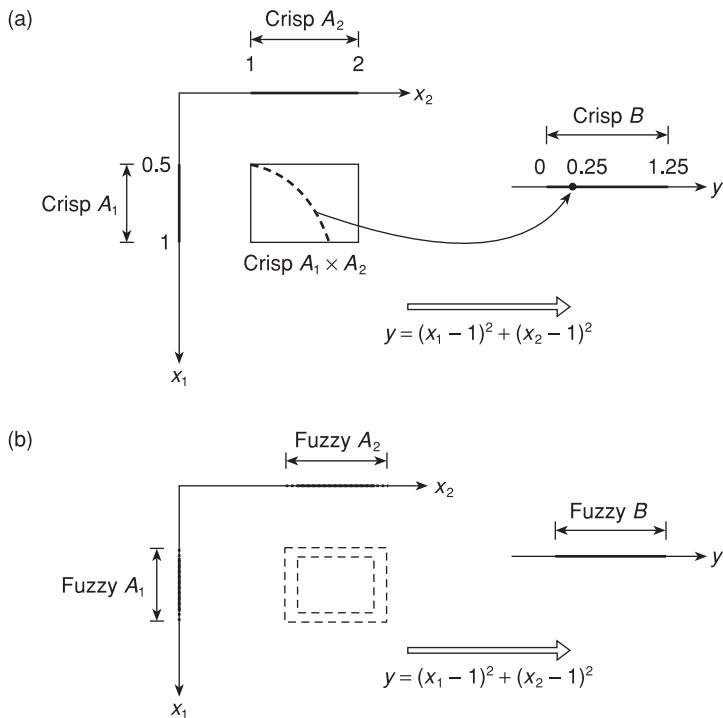
X and Y. Two separate fuzzy sets in X and Y need not be identified in this definition of a fuzzy relation. Both the Cartesian product of two fuzzy sets and the fuzzy implication are “special cases” of fuzzy relation where the membership functions of the individual fuzzy sets need to be identified in their definition. The concept of Cartesian product can be directly extended to more than two fuzzy sets.

### 2.9.3 Extension principle

The extension principle was introduced by Zadeh to provide a method for “extending” a crisp relation between standard analytical entities (crisp sets, crisp variables) to the same *crisp relation* of fuzzy entities (fuzzy sets). Consider the crisp relation  $f$  given by:

$$y = f(x_1, x_2, \dots, x_r) \quad (2.35)$$

where  $x_i$  are defined in the independent universes  $X_i$ ,  $i = 1, 2, \dots, r$ , and  $y$  is defined in the universe Y. This relation generally is a many-to-one mapping



**Figure 2.20:** A crisp mapping from a product space to a line: (a) An example of crisp sets. (b) An example of fuzzy sets (extension principle)

from the  $r$ -dimensional Cartesian product space  $X_1 \times X_2 \times \dots \times X_r$  to the one-dimensional space  $Y$ ; thus

$$f: X_1 \times X_2 \times \dots \times X_r \rightarrow Y \quad (2.36)$$

Suppose that the elements  $x_i$  are restricted to crisp subsets  $A_i$  of  $X_i$ . Then the crisp relation  $f$  maps elements  $(x_1, x_2, \dots, x_r)$  within the Cartesian product  $A_1 \times A_2 \times \dots \times A_r$  onto a crisp set  $B$ , which is a subset of  $Y$ . An example (in the continuous case) is given in Figure 2.20(a). Note that this is a many-to-one mapping. For instance, the entire quarter circle with center (1, 1) and radius 0.5 in the product space  $A_1 \times A_2$  is mapped on to the single point 0.25 on the  $Y$  line.

This idea can be extended to the case in which the sets are fuzzy while the relation  $f$  is still crisp. The *extension principle* provides the means of doing this. According to the extension principle, the fuzzy set  $B$  to which the elements  $y$  belong has a membership function given by

$$\mu_B(y) = \sup_{(x_1, x_2, \dots, x_r) = f^{-1}(y)} \{\min[\mu_{A_1}(x_1), \mu_{A_2}(x_2), \dots, \mu_{A_r}(x_r)]\} \quad (2.37)$$

Here,  $(x_1, x_2, \dots, x_r)$  denote all possible solutions of  $f^{-1}(y)$  for a particular value of  $y$ . Note that the *min* operation is applied first because a point  $(x_1, x_2, \dots, x_r)$  in the Cartesian product space is considered, each  $x_i$  having a membership grade  $\mu_{A_i}(x_i)$ , which are connected by an “AND” (see equation (2.34)). The *supremum* is applied over the mapping onto  $B$  because more than one combination of  $(x_1, x_2, \dots, x_r)$  in the fuzzy Cartesian product space  $A_1 \times A_2 \times \dots \times A_r$  may be mapped to the same element  $y$  in the fuzzy set  $B$  (i.e.,  $f^{-1}(y)$  may have multiple solutions) and these multiple possibilities are connected by “OR” operations (in other words, the most possible mapping is the one with the highest membership grade). An example of a mapping from a two-dimensional fuzzy space  $A_1 \times A_2$  onto a one-dimensional fuzzy set  $B$  (in the continuous case) is shown in Figure 2.20(b).

In the scalar (one-dimensional) case of  $x$ , a single fuzzy set  $A$  in  $X$  is related to a fuzzy set  $B$  in  $Y$ , through a crisp relation  $y = f(x)$ . Then, the extension principle is given by

$$\mu_{f(A)}(y) = \sup_{x=f^{-1}(y)} [\mu_A(x)] \quad (2.38)$$

where  $B$  is denoted by  $f(A)$ . The supremum operation is taken over the solution set of  $x = f^{-1}(y)$  for a given  $y$ .

### Example 2.21

Consider a crisp algebraic system given by

$$y = f(u)$$

where

$u$  = system input

$y$  = system output

$f$  = crisp (non-fuzzy) algebraic function, which describes the system.

Suppose that the system input is a fuzzy set  $A$  and is given by the membership function  $\mu_A(u)$ . A mathematical expression for determining the corresponding fuzzy output  $u$  with membership function  $\mu_B(y)$  is given by the extension principle (2.38), for this single-input case. If there are multiple inputs, equation (2.37) should be used.

Suppose that the system function and the fuzzy input are as shown in Figure 2.21(a). The corresponding output  $\mu_B(y)$  may be sketched by applying equation (2.38), as shown in Figure 2.21(b).

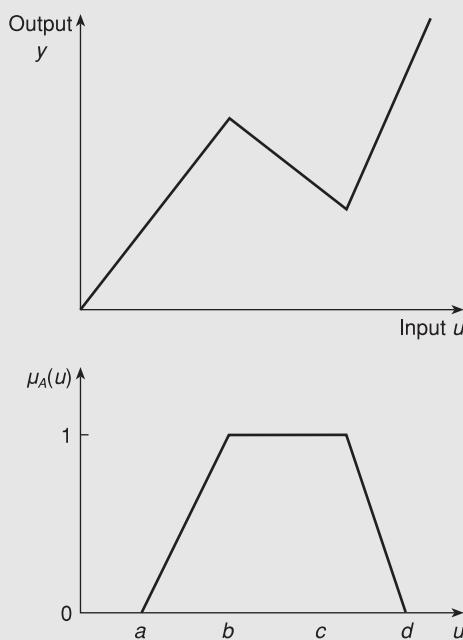


Figure 2.21(a): A non-fuzzy algebraic system with a fuzzy input

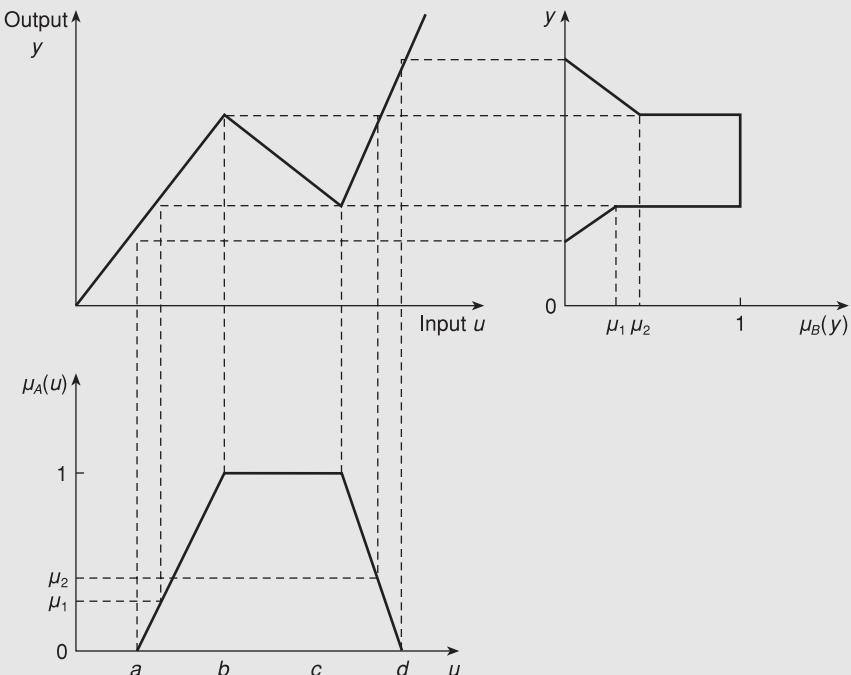


Figure 2.21(b): Application of the extension principle

The extension principle is applicable to the special case where the relation is crisp (and the connected entities are fuzzy). The *composition* operation is more general, where the relation itself is fuzzy (and the connected entities are also fuzzy). This useful operation is considered next.

## 2.10 Composition and inference

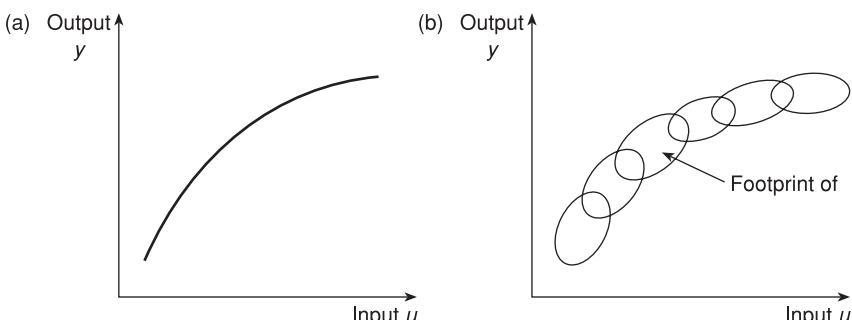
The representation theorem, as discussed in the previous section, may be interpreted as a decision-making tool where a fuzzy output is determined by applying a fuzzy input to a crisp system (a crisp or non-fuzzy relation). The example in Figure 2.21 illustrates this process. In the present section, this procedure is extended to fuzzy systems. A fuzzy system may be represented by a set of fuzzy if-then rules, which may be aggregated into a single (multi-variable) membership function – a fuzzy relation. Application of a fuzzy input to a fuzzy relation in order to infer a fuzzy output is the basis of decision-making in a fuzzy knowledge-based system. This idea is illustrated in Figure 2.22.

Decision-making using fuzzy logic is known as fuzzy inference or *approximate reasoning*. The *compositional rule of inference* is utilized for this purpose. We have already introduced the concept of *fuzzy implication*. We now introduce the terms *projection*, *cylindrical extension*, and *join*, which lead to the concept of *composition*. Finally, the compositional rule of inference is discussed, incorporating all these ideas.

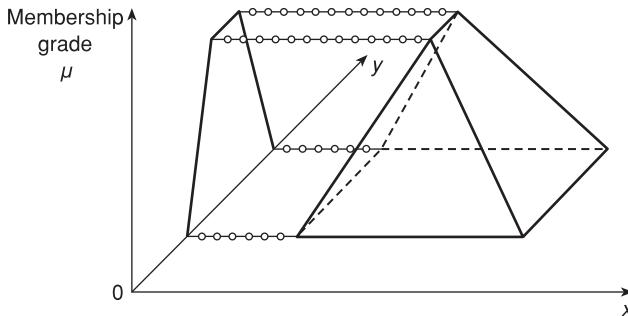
### 2.10.1 Projection

Consider a fuzzy relation  $R$  in the  $n$ -dimensional Cartesian product space  $X_1 \times X_2 \times \dots \times X_n$ . Without loss of generality, suppose that the  $n$  indices (coordinates) are renumbered (rearranged) as follows:

$$\{1, 2, \dots, n\} \rightarrow \{i_1, i_2, \dots, i_r, j_1, j_2, \dots, j_m\} \quad (2.39)$$



**Figure 2.22:** (a) Fuzzy decision-making using a crisp relation (extension principle).  
(b) Fuzzy decision-making using a fuzzy relation (composition)



**Figure 2.23:** An example of fuzzy projection

Note that  $r + m = n$  and that  $i$  and  $j$  denote the first  $r$  and the second  $m$  sets, respectively, of newly assigned  $n$  indices. The projection of  $R$  on the  $r$ -dimensional subspace  $X_{i_1} \times X_{i_2} \times \dots \times X_{i_r}$  is denoted by

$$\text{Proj}[R; X_{i_1} \times X_{i_2} \times \dots \times X_{i_r}]$$

This is an  $r$ -dimensional fuzzy relation  $P$  and its membership function is given by

$$\mu_P(x_{i_1}, x_{i_2}, \dots, x_{i_r}) = \sup_{x_{j_1}, x_{j_2}, \dots, x_{j_m}} [\mu_R(x_1, x_2, \dots, x_n)] \quad (2.40)$$

The rationale for using the supremum operation on the membership function of  $R$  should be clear in view of the fact that we have a many-to-one mapping from an  $n$ -dimensional space to an  $r$ -dimensional space, with  $r < n$ , and the most possible one among them will govern the possibility of the overall result.

As an example, consider the Cartesian product  $A_1 \times A_2$  of two fuzzy sets  $A_1$  and  $A_2$ , as shown in Figure 2.19(b), which is a fuzzy relation (continuous) in the two-dimensional space  $X_1 \times X_2$ . The fuzzy projection of  $R$  on  $X_1$  has a membership function  $\mu_{A_1}(x_1)$  which is exactly the geometric projection (shadow) of  $\mu_R(x_1, x_2)$  on the  $\mu - x_1$  plane. Similarly, the fuzzy projection of  $R$  on  $X_2$  has a membership function  $\mu_{A_2}(x_2)$  which is exactly the geometric projection (shadow) of  $\mu_R(x_1, x_2)$  on the  $\mu - x_2$  plane, as is clear from the figure. Another example is shown in Figure 2.23, where a two-variable membership function is projected on to the plane of one of the variables. Again, the application of the *sup* operation to determine the projection is clear.

### Example 2.22

Consider an  $n$ -ary fuzzy relation  $R(x, y, z, \dots)$ , which has  $n$  independent variables (coordinate axes),  $x, y, z, \dots$ . How many different projections are possible for this relation, in various subspaces? Explain how you arrived at this number.

Consider a discrete ternary ( $n = 3$ ) fuzzy relation (rule base) given by the following “pages” of binary  $(x_i, y_j)$  relation along the  $z$  axis:

$$R_{z_1}(x_i, y_j) = \begin{bmatrix} 0.5 & 0.4 & 0.2 \\ 0.3 & 0.8 & 0.5 \\ 0.1 & 0.6 & 0.6 \end{bmatrix}; \quad R_{z_2}(x_i, y_j) = \begin{bmatrix} 0.6 & 0.5 & 0.3 \\ 0.4 & 1.0 & 0.6 \\ 0.2 & 0.6 & 0.8 \end{bmatrix};$$

$$R_{z_3}(x_i, y_j) = \begin{bmatrix} 0.4 & 0.3 & 0.1 \\ 0.2 & 0.7 & 0.3 \\ 0.0 & 0.5 & 0.5 \end{bmatrix}$$

Determine all possible projections of this relation.

### **Solution**

From  $n$ -space to  $n - 1$  space there are  $n$  projections. From  $n$ -space to  $n - 2$  space, there are a total of  $n(n - 1)$  projections, but only half of them are distinct because the other half arrive at the same projections but through a different order of sequences of projections. Hence, only  $\frac{n(n - 1)}{2!}$  projections of these are distinct. Similarly, from  $n$ -space to  $n - r$  space, there are only  $\frac{n(n - 1) \dots (n - r)}{r!}$  distinct projections. Hence, the total number of distinct projections is:

$$S_n = n + \frac{n(n - 1)}{2!} + \frac{n(n - 1)(n - 2)}{3!} + \dots + \frac{n(n - 1)(n - 2) \dots \times 3 \times 2 \times 1}{(n - 1)!}$$

Now consider the given three-dimensional rule base  $R(x_i, y_j, z_k)$ . Projection on  $X \times Y$  is obtained by taking the element-wise maximum of the given three matrices; thus:

$$\text{Proj } R(x_i, y_j, z_k) = \max_{z_k} R(x_i, y_j, z_k) = x_i \begin{bmatrix} \rightarrow y_j \\ 0.6 & 0.5 & 0.3 \\ 0.4 & 1.0 & 0.6 \\ 0.2 & 0.6 & 0.8 \end{bmatrix}$$

This result can now be projected onto  $Y$  by taking the maximum of the elements in each column; and similarly projected onto  $X$  by taking the maximum of the elements in each row; thus:

$$\text{Proj } R(x_i, y_j, z_k) = [0.6 \quad \stackrel{\rightarrow y_j}{1.0} \quad 0.8]$$

$$\text{Proj } R(x_i, y_j, z_k) = x_i \begin{bmatrix} 0.6 \\ 1.0 \\ 0.8 \end{bmatrix}$$

Projection on  $Y \times Z$  is obtained by first taking the element-wise maximum of each column of the given three matrices, and then assembling the resulting rows into a matrix; thus:

$$\text{Proj } R(x_i, y_j, z_k) = \max_{x_i} R(x_i, y_j, z_k) = z_k \begin{bmatrix} \rightarrow y_j \\ 0.5 & 0.8 & 0.6 \\ 0.6 & 1.0 & 0.8 \\ 0.4 & 0.7 & 0.5 \end{bmatrix}$$

From this result, the projection on  $Y$  is obtained by taking the maximum of the elements in each column; and similarly projection on  $Z$  is obtained by taking the maximum of the elements in each row; thus:

$$\text{Proj } R(x_i, y_j, z_k) = [0.6 \quad 1.0 \quad 0.8] \\ \downarrow y_j \\ \text{Proj } R(x_i, y_j, z_k) = z_k \begin{bmatrix} 0.8 \\ 1.0 \\ 0.7 \end{bmatrix}$$

Note that the projection on  $Y$  obtained in this manner is identical to what we obtained previously, as it should be.

Projection on  $Z \times X$  is obtained by first taking the element-wise maximum of each row of the given three matrices, and then assembling the resulting columns into a matrix; thus:

$$\text{Proj } R(x_i, y_j, z_k) = \max_{y_j} R(x_i, y_j, z_k) = x_i \begin{bmatrix} \rightarrow z_k \\ 0.5 & 0.6 & 0.4 \\ 0.8 & 1.0 & 0.7 \\ 0.6 & 0.8 & 0.5 \end{bmatrix}$$

From this result, the projection on  $Z$  is obtained by taking the maximum of the elements in each column; and similarly projection on  $X$  is obtained by taking the maximum of the elements in each row; thus:

$$\text{Proj } R(x_i, y_j, z_k) = [0.8 \quad 1.0 \quad 0.7] \\ \downarrow z_k \\ \text{Proj } R(x_i, y_j, z_k) = x_i \begin{bmatrix} 0.6 \\ 1.0 \\ 0.8 \end{bmatrix}$$

Again, note that these projections on  $Z$  and  $X$  are identical to what we obtained previously, as they should be.

In summary, only three of the six projections onto a single axis (1-D) are distinct. Hence, there are a total of only six distinct projections (three onto 2-D spaces and three onto 1-D spaces).

## 2.10.2 Cylindrical extension

Consider the Cartesian product space  $X_1 \times X_2 \times \dots \times X_n$ , and suppose that the  $n$  indices are rearranged (renumbered) as follows:

$$\{1, 2, \dots, n\} \rightarrow \{i_1, i_2, \dots, i_r, j_1, j_2, \dots, j_m\}$$

Again note that  $r + m = n$  and that  $i$  and  $j$  denote the first  $r$  and the second  $m$  sets, respectively, of the newly assigned  $n$  indices. Now consider a fuzzy relation  $R$  defined in the subspace  $X_{i1} \times X_{i2} \times \dots \times X_{ir}$ . Its *cylindrical extension*, denoted by  $C(R)$ , into the entire  $n$ -dimensional space is given by

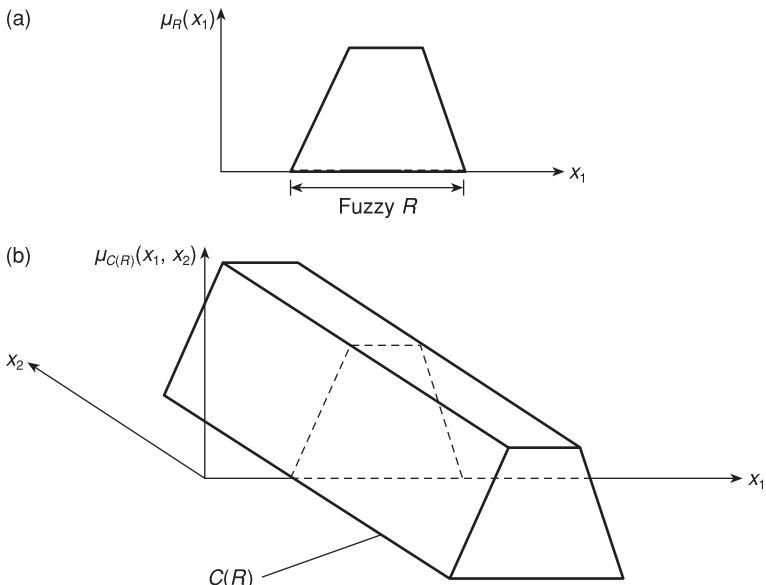
$$C(R) = \int_{X_1 \times X_2 \times \dots \times X_n} \frac{\mu_R(x_{i1}, x_{i2}, \dots, x_{ir})}{x_1, x_2, \dots, x_n} \quad (2.41)$$

Specifically, the membership value is maintained constant along each of the extended ( $m$ ) axes. Hence, the membership function of the cylindrical extension is

$$\mu_{C(R)}(x_1, x_2, \dots, x_n) = \mu_R(x_{i1}, x_{i2}, \dots, x_{ir}), \quad \forall x_{j1}, x_{j2}, \dots, x_{jm} \quad (2.41)^*$$

It should also be clear that  $C(R)$  is obtained by the Cartesian product of  $R$  with  $X_{j1} \times X_{j2} \times \dots \times X_{jm}$ . That is, the membership function  $\mu_{C(R)}(x_1, x_2, \dots, x_n)$  is obtained by performing the *min* operation on  $\mu_R(x_{i1}, x_{i2}, \dots, x_{ir})$  and a unity membership function defined in the subspace  $X_{j1} \times X_{j2} \times \dots \times X_{jm}$  (i.e.,  $\mu_{X_{j1} \times X_{j2} \times \dots \times X_{jm}}$ ). Note that a cylindrical extension is a fuzzy set in the  $n$ -dimensional space.

Cylindrical extension may be interpreted as the converse (reverse) operation of projection. An example is given in Figure 2.24. Here, a fuzzy set (or



**Figure 2.24:** (a) A fuzzy relation (fuzzy set). (b) Its cylindrical extension

fuzzy relation)  $R$  defined in the universe  $X_1$  has been cylindrically extended to a fuzzy relation in the Cartesian space  $X_1 \times X_2$ .

### Example 2.23

A discrete fuzzy relation  $R(x_i, y_j)$  is given by the following membership function matrix, defined in  $X \times Y$  where  $X = [0, 1, 2, 3, 4]$  and  $Y = [0, 1, 2, 3, 4]$ :

	$y_0 = 0$	$y_1 = 1$	$y_2 = 2$	$y_3 = 3$	$y_4 = 4$
$x_0 = 0$	0.0	0.4	0.7	0.3	0.0
$x_1 = 1$	0.1	0.5	0.8	0.4	0.1
$x_2 = 2$	0.6	0.7	1.0	0.5	0.2
$x_3 = 3$	0.3	0.4	0.9	0.7	0.4
$x_4 = 4$	0.0	0.1	0.5	0.3	0.1

The following discrete fuzzy sets are derived from  $R(x_i, y_j)$ :

$$A(x_i) = \text{Projection } R(x_i, y_j)$$

$x_i$

$$B(y_j) = \text{Projection } R(x_i, y_j)$$

$y_j$

$$A_1(x_i) = R(x_i, 1)$$

$$A_2(x_i) = R(x_i, 2)$$

- (a) Determine and sketch the (discrete) membership functions of the following fuzzy sets and fuzzy relations:

- |                     |                                                                   |
|---------------------|-------------------------------------------------------------------|
| (i) $A(x_i)$        | (vii) $\alpha$ -cut of $A_1$ for $\alpha = 0.2, 0.42$ , and $0.5$ |
| (ii) $A_1(x_i)$     | (viii) $A \rightarrow B$                                          |
| (iii) $A_2(x_i)$    | (ix) $A \times B$                                                 |
| (iv) $B(y_j)$       | (x) Cylindrical extension of $A$ in $X \times Y$                  |
| (v) $A_1 \cup A_2$  | (xi) Cylindrical extension of $B$ in $X \times Y$                 |
| (vi) $A_1 \cap A_2$ |                                                                   |

- (b) The fuzzy relation  $R(x_i, y_j)$  in  $X \times Y$  is to be mapped to a fuzzy set  $C(z_k)$  in  $Z$ , using the crisp function  $z = x + y$ . Determine and sketch  $C$ .

**Solution**

(a)

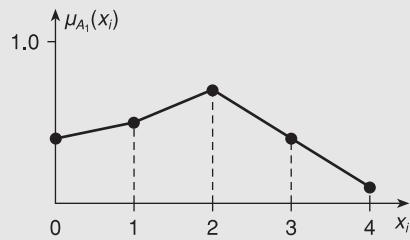
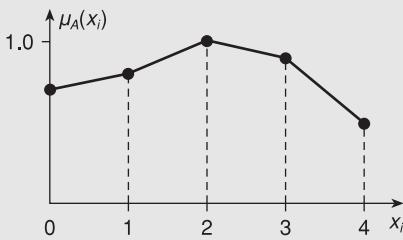
(i)  $A(x_i) = \text{Projection}_{x_i} R(x_i, y_j)$

$$\mu_A(x_i) = \sup_{y_j} [\mu_R(x_i, y_j)]$$

$$A(x_i) = \left[ \frac{0.7}{0}, \frac{0.8}{1}, \frac{1.0}{2}, \frac{0.9}{3}, \frac{0.5}{4} \right]$$

(ii)  $A_1(x_i) = R(x_i, 1)$

$$A_1(x_i) = \left[ \frac{0.4}{0}, \frac{0.5}{1}, \frac{0.7}{2}, \frac{0.4}{3}, \frac{0.1}{4} \right]$$

**Figure 2.25:** Discrete membership functions of various fuzzy sets and fuzzy relations

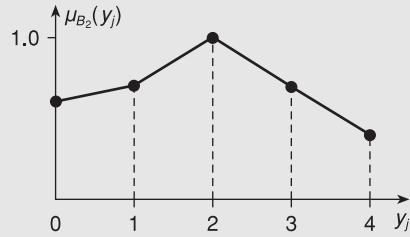
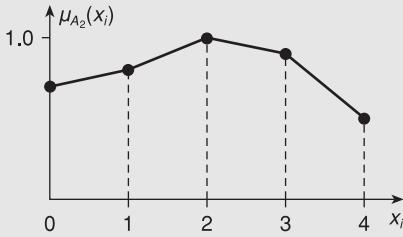
(iii)  $A_2(x_i) = R(x_i, 2)$

$$A_2(x_i) = \left[ \frac{0.7}{0}, \frac{0.8}{1}, \frac{1.0}{2}, \frac{0.9}{3}, \frac{0.5}{4} \right]$$

(iv)  $B(y_j) = \text{Projection}_{y_j} R(x_i, y_j)$

$$\mu_B(y_j) = \sup_{x_i} \{\mu_R(x_i, y_j)\}$$

$$B(y_j) = \left[ \frac{0.6}{0}, \frac{0.7}{1}, \frac{1.0}{2}, \frac{0.7}{3}, \frac{0.4}{4} \right]$$

**Figure 2.25:** (cont'd)

(v)  $A_1 \cup A_2$

$$\mu_{A_1 \cup A_2}(x_i) = \max\{\mu_{A_1}(x_i), \mu_{A_2}(x_i)\}$$

$$A_1 \cup A_2 = \left[ \frac{0.7}{0}, \frac{0.8}{1}, \frac{1.0}{2}, \frac{0.9}{3}, \frac{0.5}{4} \right]$$

(vi)  $A_1 \cap A_2$

$$\mu_{A_1 \cap A_2}(x_i) = \min\{\mu_{A_1}(x_i), \mu_{A_2}(x_i)\}$$

$$A_1 \cap A_2 = \left[ \frac{0.4}{0}, \frac{0.5}{1}, \frac{0.7}{2}, \frac{0.4}{3}, \frac{0.1}{4} \right]$$

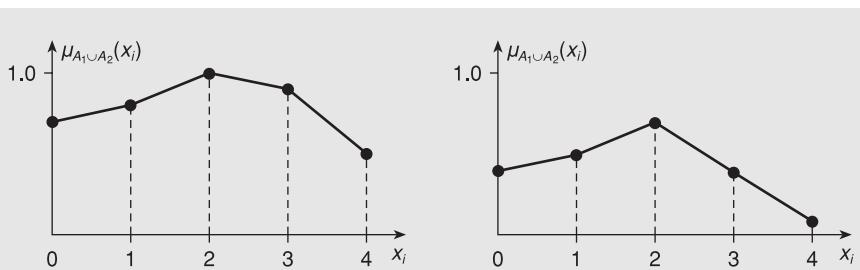
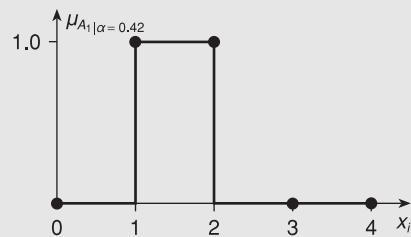
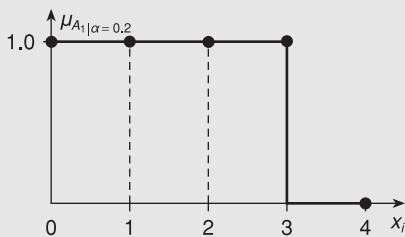


Figure 2.25: (cont'd)

$$\text{(vii)} \quad \begin{aligned} \mu_{F\alpha}(x) &= 1 \text{ for } \mu_F(x) \geq \alpha \\ &= 0 \text{ for } \mu_F(x) < \alpha \end{aligned}$$

$$\text{(a)} \quad A_{1\alpha|\alpha=0.2} = \left[ \frac{1.0}{0}, \frac{1.0}{1}, \frac{1.0}{2}, \frac{1.0}{3}, \frac{0.0}{4} \right]$$

$$\text{(b)} \quad A_{1\alpha|\alpha=0.42} = \left[ \frac{0.0}{0}, \frac{1.0}{1}, \frac{1.0}{2}, \frac{0.0}{3}, \frac{0.0}{4} \right]$$



$$\text{(c)} \quad A_{1\alpha|\alpha=0.5} = \left[ \frac{0.0}{0}, \frac{1.0}{1}, \frac{1.0}{2}, \frac{0.0}{3}, \frac{0.0}{4} \right]$$

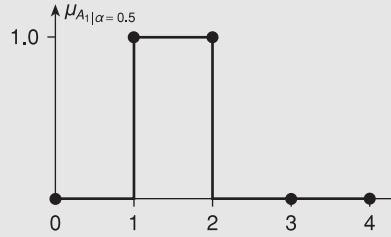


Figure 2.25: (cont'd)

$$\text{(viii)} \quad \mu_{A \rightarrow B}(x_i, y_j) = \min \{ \mu_A(x_i), \mu_B(y_j) \}$$

	$y_0$	$y_1$	$y_2$	$y_3$	$y_4$
$x_0$	0.6	0.7	0.7	0.7	0.4
$x_1$	0.6	0.7	0.8	0.7	0.4
$A \rightarrow B = x_2$	0.6	0.7	1.0	0.7	0.4
$x_3$	0.6	0.7	0.9	0.7	0.4
$x_4$	0.5	0.5	0.5	0.5	0.4

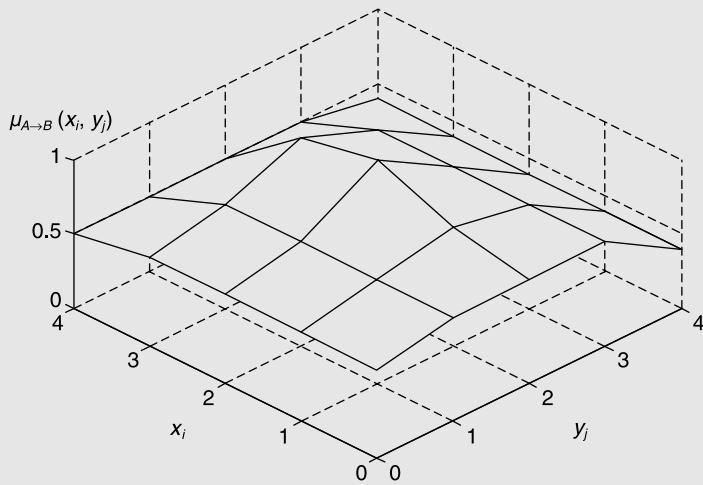


Figure 2.25: (cont'd)

(ix)  $A \times B$  (same as (viii)).

$$(x) \quad C(A) = \sum_{x_i, y_j} \frac{\mu_A(x_i)}{x_i, y_j} = \begin{matrix} & y_0 & y_1 & y_2 & y_3 & y_4 \\ x_0 & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 \\ x_1 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 \\ x_2 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 \\ x_3 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 \\ x_4 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \end{matrix}$$

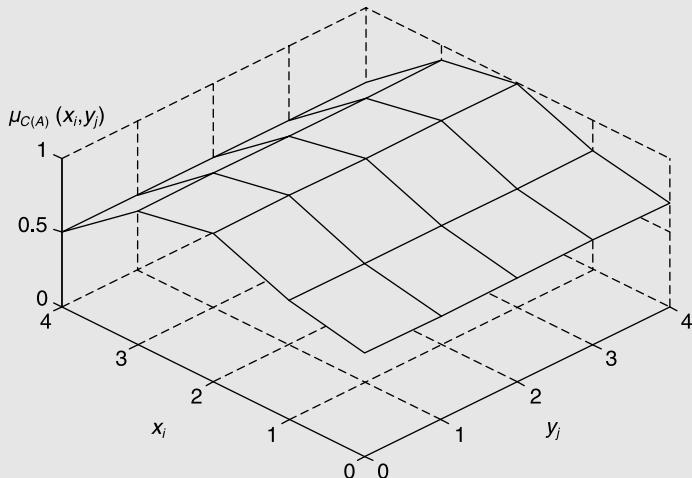


Figure 2.25: (cont'd)

$$(xi) \quad C(B) = \sum_{X \times Y} \frac{\mu_B(y_i)}{x_i, y_i} = \begin{matrix} & y_0 & y_1 & y_2 & y_3 & y_4 \\ x_0 & 0.6 & 0.7 & 1.0 & 0.7 & 0.4 \\ x_1 & 0.6 & 0.7 & 1.0 & 0.7 & 0.4 \\ x_2 & 0.6 & 0.7 & 1.0 & 0.7 & 0.4 \\ x_3 & 0.6 & 0.7 & 1.0 & 0.7 & 0.4 \\ x_4 & 0.6 & 0.7 & 1.0 & 0.7 & 0.4 \end{matrix}$$

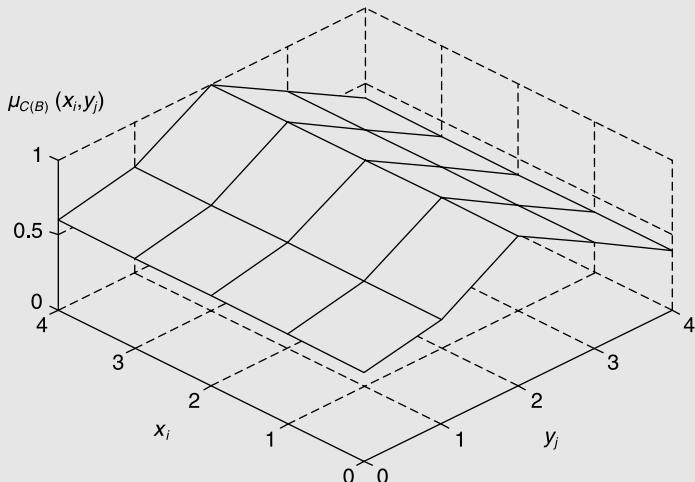


Figure 2.25: (cont'd)

(b) The extension principle is applied, where the crisp relation  $z = x + y$  is used.

We get

$$\mu_C(z_k) = \sup_{z=x+y} [\mu_R(x_i, y_j)]$$

The fuzzy relation  $R(x_i, y_j)$  in the  $X \times Y$  domain is mapped to  $C(z_k)$  in the  $Z$  domain, as shown in Figure 2.26. Note that the diagonal broken lines are the  $x + y = z = \text{constant}$  lines, for different feasible values of  $z$ . We get

$$C(z_k) = \left\{ \frac{0.0}{0}, \frac{0.4}{1}, \frac{0.7}{2}, \frac{0.8}{3}, \frac{1.0}{4}, \frac{0.9}{5}, \frac{0.7}{6}, \frac{0.4}{7}, \frac{0.1}{8} \right\}$$

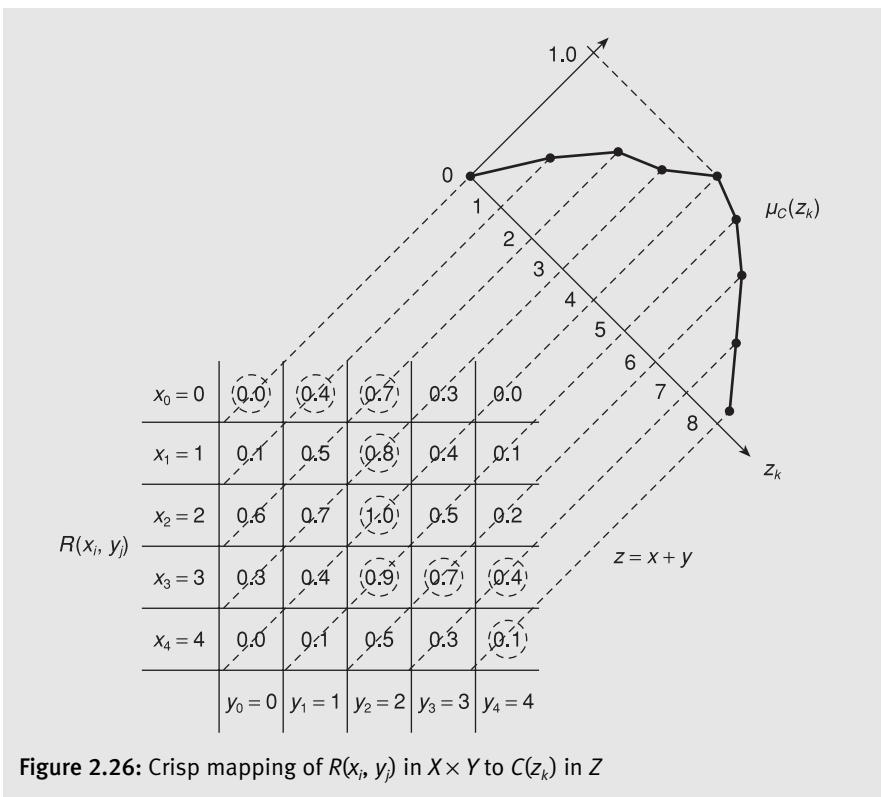


Figure 2.26: Crisp mapping of  $R(x_i, y_j)$  in  $X \times Y$  to  $C(z_k)$  in  $Z$

### 2.10.3 Join

Consider a fuzzy relation  $R$  in the  $r$ -dimensional subspace  $X_1 \times X_2 \times \dots \times X_r$ , and a second fuzzy relation  $S$  in the  $(n - m + 1)$ -dimensional subspace  $X_m \times X_{m+1} \times \dots \times X_n$ , such that  $m < r + 2$  (i.e., the union of the two subspaces  $R$  and  $S$  should form the original  $n$ -dimensional space  $X_1 \times X_2 \times \dots \times X_n$ ). The join of the fuzzy relations (or fuzzy sets)  $R$  and  $S$  is a fuzzy relation (or fuzzy set) in the overall  $X_1 \times X_2 \times \dots \times X_n$  space, and is given by the intersection of their cylindrical extensions; thus:

$$\text{Join}(R, S) = C(R) \wedge C(S) \quad \text{in } X_1 \times X_2 \times \dots \times X_n \quad (2.42)$$

Its membership function is given by

$$\mu_{\text{Join}(R,S)}(x_1, x_2, \dots, x_n) = \min[\mu_{C(R)}(x_1, x_2, \dots, x_n), \mu_{C(S)}(x_1, x_2, \dots, x_n)] \quad (2.43)$$

Note that the operation  $\min$  is applied here because the intersection of two fuzzy sets is considered.

## 2.10.4 Composition

Consider a fuzzy relation (fuzzy set)  $R$  in the  $r$ -dimensional subspace  $X_1 \times X_2 \times \dots \times X_r$  and a second fuzzy relation (fuzzy set)  $S$  in the  $(n - m + 1)$ -dimensional subspace  $X_m \times X_{m+1} \times \dots \times X_n$  such that  $m < r + 1$ . Note that unlike the previous case of join, the two subspaces are never disjoint, and hence their intersection is never null (i.e., there is at least one common dimension in the subspaces  $R$  and  $S$ ). As before, however, the union of the two subspaces gives the overall  $n$ -dimensional space  $X_1 \times X_2 \times \dots \times X_n$ . The *composition* of  $R$  and  $S$  is denoted by  $R \circ S$  and is given by

$$R \circ S = \text{Proj}[\text{Join}(R, S); X_1, \dots, X_{m-1}, X_{r+1}, \dots, X_n] \quad (2.44)$$

Specifically, we take the join of the two sets  $R$  and  $S$ , as given by equation (2.42), and then project the resulting fuzzy set (in the  $n$ -dimensional space) onto the subspace formed by the disjoint parts of the two subspaces in which the fuzzy sets  $R$  and  $S$  are defined. The membership function of the resulting fuzzy set is obtained from the membership functions of  $R$  and  $S$ , while noting that the operation *min* applies for *join* and the operation *supremum* applies for *projection*, as discussed before. Specifically,

$$\mu_{(R \circ S)} = \sup_{x_m, \dots, x_r} [\min(\mu_R, \mu_S)] \quad (2.45)$$

This is called the *sup-min composition*.

Composition can be interpreted as a matching of the common (known) parts of two fuzzy sets, and then making an inference about the disjoint (unknown) parts according to the result of the matching process. Specially, the two sets are combined and matched (through *join*) over their common subspace. The result is then projected (through *supremum*) over to the inference subspace (non-overlapping subspace), giving a fuzzy set defined over the disjoint portions of the two subspaces. This process of matching is quite analogous to matching of data with the condition part of a rule in the conventional rule-based decision-making. In this regard composition plays a crucial role in fuzzy inference (fuzzy decision-making) and fuzzy knowledge-based systems.

As a simple example, consider a fuzzy relation  $R$  defined in the  $X \times Y$  space and another fuzzy relation  $S$  defined in the  $Y \times Z$  space. The composition of these two fuzzy relations is given by

$$\mu_{(R \circ S)}(x, z) = \sup_{y \in Y} \{\min(\mu_R(x, y), \mu_S(y, z))\} \quad (2.46)$$

which is defined in the  $X \times Z$  space.

### 2.10.4.1 Sup-product composition

In equation (2.45) if we use the “product” operation in place of “min” we get the *sup-product composition* given by

$$\mu_{(R \bullet S)} = \sup_{x_m, \dots, x_r} [\mu_R \times \mu_S] \quad (2.47)$$

This composition is denoted by “ $\bullet$ ” and is also known as the *sup-dot composition*. In fact, any T-norm may be used to represent the matching (join) operation in composition, in place of min or product. This is termed *sup-T composition*. Note that, in general, composition is a *commutative* operation:  $R \circ S = S \circ R$ , because the T-norm operation is commutative ( $aTb = bTa$ ).

In summary, the composition operation involves:

- (1) Matching of data and a knowledge base, using the join (or T-norm) operation
- (2) Making an inference (decision) on the variables in the subspace of the knowledge base that is outside the subspace of data (i.e., non-overlapping subspace), using the projection (sup) operation.

### 2.10.5 Compositional rule of inference

In knowledge-based systems, knowledge is often expressed as rules of the form:

“IF condition  $Y_1$  is  $y_1$  AND IF condition  $Y_2$  is  $y_2$  THEN action  $C$  is  $c$ .”

In fuzzy knowledge-based systems (e.g., fuzzy control systems), rules of this type are linguistic statements of expert knowledge in which  $y_1$ ,  $y_2$ , and  $c$  are fuzzy quantities (e.g., small negative, fast, large positive). These rules are fuzzy relations that employ the fuzzy implication (IF-THEN). The collective set of fuzzy relations forms the knowledge base of the fuzzy system. Let us denote the fuzzy relation formed by this collection of rules as the fuzzy set  $K$ . This relation is an aggregation of the individual rules, and may be represented by a multivariable membership function. In a fuzzy decision-making process (e.g., in fuzzy logic control), the rule base (knowledge base)  $K$  is first collectively matched with the available data (context). Next, an inference is made on another fuzzy variable that is represented in the knowledge base, on this basis. The matching and inference-making are done using the composition operation, as discussed previously. The application of composition to make inferences in this manner is known as the *compositional rule of inference* (CRI).

Suppose that the available data (context) are denoted by a fuzzy relation (fuzzy set)  $D$  and the inference (action) is denoted by a fuzzy relation (fuzzy set)  $I$ . The compositional rule of inference states that:

$$I = D \circ K \quad (2.48)$$

By using equation (2.48) we can determine the membership function of the inference (decision, action)  $I$  by matching the membership functions of data  $D$  and the knowledge base  $K$ . In other words, the compositional rule of inference makes inferences using the composition operation. Specifically, we have:

$$\mu_I = \sup_Y \min [\mu_D, \mu_K] \quad (2.49)$$

This inference is the output of the knowledge-based decision-making system.

Note that  $Y$  denotes the space in which the data  $D$  are defined, and it is a subspace of the space in which the rule base  $K$  is defined. The knowledge base  $K$  consists of rules containing AND connectives and fuzzy implication (IF-THEN), both of which can be represented by *min* operation. Hence, the membership function of each rule can be formed by the membership functions of the constituent variables by applying *min*. Next, since the individual rules are connected by an OR (Else) operation, which can be represented by *max* operation, the overall membership function of  $K$  may be determined by applying *max* to the membership functions of the individual rules.

For example, consider a control system. Usually the context would be the measured outputs  $Y$  of the process. The control action that drives the process is  $C$ . Typically, both these variables are crisp, but let us ignore this fact for the time being and assume them to be fuzzy, for general consideration. Suppose that the control knowledge base is denoted by  $R$ , a fuzzy relation. The method of obtaining the rule base  $R$  is analogous to model identification in conventional crisp control. Then, by applying the compositional rule of inference we get the fuzzy control action as:

$$\mu_C = \max_Y \min(\mu_Y, \mu_R) \quad (2.48)^*$$

### Example 2.24

Suppose that a fuzzy set  $A$  represents the output of a process and that it belongs to a discrete and finite universe  $Y$  of cardinality (number of elements) 5. Furthermore, suppose that a fuzzy set  $C$  represents the control input to the process and that it belongs to a discrete and finite universe  $Z$  of cardinality 4. It is given that:

$$\begin{aligned} A &= 0.2/y_2 + 1.0/y_3 + 0.8/y_4 + 0.1/y_5 \\ C &= 0.1/z_1 + 0.7/z_2 + 1.0/z_3 + 0.4/z_4. \end{aligned}$$

A fuzzy relation  $R$  is defined by the fuzzy implication  $A \rightarrow C$ . The membership function of  $R$  is obtained by applying *min* operation to  $A$  and  $C$ ; thus:

$$\mu_R(y_i, z_j) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0.1 & 0.2 & 0.2 & 0.2 \\ 0.1 & 0.7 & 1.0 & 0.4 \\ 0.1 & 0.7 & 0.8 & 0.4 \\ 0.1 & 0.1 & 0.1 & 0.1 \end{bmatrix}$$

which is defined in the two-dimensional space  $X \times Y$ . Note that the rows correspond to  $y_i$  and the columns correspond to  $z_j$ . This matrix ( $R$ ) represents the rule base in fuzzy decision-making (fuzzy logic control). Specifically, the

information carried by the fuzzy rules  $A \rightarrow C$  has been reduced to a relation  $R$  in the form of a matrix. Note that  $R$  is not a decision table for fuzzy control, even though a decision table can be derived using  $R$  by successively applying the compositional rule of inference to  $R$  for expected process responses.

Now suppose that a process measurement  $y_0$  is made and that the measurement is closest to the element  $y_4$  in  $Y$ . This crisp set may be represented by a *fuzzy singleton*  $A_0$  having the membership function:

$$\mu_{A_0}(y_i) = [0, 0, 0, 0.8, 0]$$

The membership function of the corresponding fuzzy control inference  $C'$  is obtained using the CRI (equation (2.48)). Specifically, we have:

$$\begin{aligned} \mu_{C'}(z_i) &= \max_{\text{row}} \left[ \min_{\text{column}} \left[ \left( \begin{array}{c} 0 \\ 0 \\ 0 \\ 0.8 \\ 0 \end{array} \right), \left( \begin{array}{cccc} 0 & 0 & 0 & 0 \\ 0.1 & 0.2 & 0.2 & 0.2 \\ 0.1 & 0.7 & 1.0 & 0.4 \\ 0.1 & 0.7 & 0.8 & 0.4 \\ 0.1 & 0.1 & 0.1 & 0.1 \end{array} \right) \right] \right] \\ &= \max_{\text{row}} \left[ \begin{array}{cccc} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0.1 & 0.7 & 0.8 & 0.4 \\ 0 & 0 & 0 & 0 \end{array} \right] \\ &= [0.1, 0.7, 0.8, 0.4] \end{aligned}$$

Hence, the fuzzy inference is given by

$$\mu_{C'}(z_i) = [0.1, 0.7, 0.8, 0.4]$$

This is simply the fourth row of  $\mu_R$ . This fuzzy inference must be defuzzified (i.e., made crisp) for use in process control, for example using the *centroid method*, as discussed in Chapter 3.

### 2.10.5.1 Composition through matrix multiplication

We have seen that when a membership function is discrete (i.e., when there is a finite number of elements in the corresponding universe), it can be represented as a vector of membership values. Furthermore, in the case of discrete membership functions, if the rules in a rule base have only one condition variable and one action variable, then the membership function of each rule and hence the overall (aggregated) membership function of the rule base can be represented as a matrix. In this case it is convenient to consider the

*composition* operation as a *generalized matrix multiplication* of the membership function vector of data (condition) and the membership function matrix of the rule base.

For the sup-min composition, it is clear that in the generalized matrix multiplication, the element “multiplication” has to be replaced by the “min” operation, and the element “addition” operation has to be replaced by “sup.”

For the sup-dot composition, the element “multiplication” in the matrix multiplication is retained, while the element “addition” is replaced by “sup.”

Generally, when the T-norm is used to represent the matching operation (intersection) in composition, the element “multiplication” in matrix multiplication has to be replaced by the T-norm operation, and element “addition” operation has to be replaced by “sup.” Note further that, when the membership functions are discrete, the operation *sup* is identical to the operation *max*. Then the terms *sup-T composition* and *max-T composition* mean the same thing, and similarly for *max-min composition* and *max-dot composition*.

### Example 2.25

In the previous example, the data (condition) vector is: [0, 0, 0, 0.8, 0] and the knowledge base is:

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0.1 & 0.2 & 0.2 & 0.2 \\ 0.1 & 0.7 & 1.0 & 0.4 \\ 0.1 & 0.7 & 0.8 & 0.4 \\ 0.1 & 0.1 & 0.1 & 0.1 \end{bmatrix}$$

Then, for the sup-min composition, the matrix multiplication is carried out as:

$$[0 \ 0 \ 0 \ 0.8 \ 0] \circ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.1 & 0.7 & 0.8 & 0.4 \\ 0 & 0 & 0 & 0 \end{bmatrix} = [0.1, \ 0.7, \ 0.8, \ 0.4]$$

For the sup-dot composition, the matrix multiplication is carried out as:

$$[0 \ 0 \ 0 \ 0.8 \ 0] \bullet \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.1 & 0.7 & 0.8 & 0.4 \\ 0 & 0 & 0 & 0 \end{bmatrix} = [0.08, \ 0.56, \ 0.64, \ 0.32]$$

## 2.10.6 Properties of composition

We have noted that the commonly used sup-min composition is a special case of the sup-T composition where the min operation is used to represent the T-norm. Similarly, in the sup-dot composition, the product operation is used as the T-norm. Hence, it is appropriate to discuss the properties of sup-T composition, since the properties of other specific compositions will follow from the general results.

### 2.10.6.1 Sup-T composition

Consider two fuzzy relations (fuzzy sets)  $P$  and  $R$  represented by the two-variable membership functions  $\mu_P(x, z)$  and  $\mu_R(z, y)$ . The sup-T composition  $P \circ R$  of  $P$  and  $R$  is given by its membership function

$$\mu_{P \circ R}(x, y) = \sup_{z \in Z} [\mu_P(x, z) \text{ } T \mu_R(z, y)] \quad (2.50)$$

Clearly this definition can be generalized to relations having more than two variables.

### 2.10.6.2 Inf-S composition

Consider two fuzzy relations (fuzzy sets)  $P$  and  $R$  represented by the two-variable membership functions  $\mu_P(x, z)$  and  $\mu_R(z, y)$ . The inf-S composition  $P \otimes R$  of  $P$  and  $R$  is given by its membership function

$$\mu_{P \otimes R}(x, y) = \inf_{z \in Z} [\mu_P(x, z) \text{ } S \mu_R(z, y)] \quad (2.51)$$

where inf represents the infimum operation (global minimum, searched piecewise continuously in the defined space), and  $S$  represents the S-norm, as usual (see Table 2.2). Again, this definition can be generalized to relations having more than two variables.

The inf-S composition is the *complement* of the sup-T composition. It may be interpreted as performing the following two steps:

- (1) Blend the data  $P$  with the knowledge base  $R$  by reinforcing the validity (membership) of the knowledge in the overlapping region ( $Z$ ) and projecting to the non-overlapping region ( $X \times Y$ ), using the S-norm (OR operation).
- (2) Scan the common region ( $Z$ ) and compress its validity (membership) to the global minimum in that region, leaving behind knowledge in the non-overlapping region ( $X \times Y$ ).

The *inf-max composition* is a special case of the inf-S composition.

### 2.10.6.3 Commutativity

The sup-T composition is commutative; thus

$$P \circ R = R \circ P \quad (2.52)$$

This holds because the T-norm operation in equation (2.50) is commutative ( $aTb = bTa$ ).

Similarly, the inf-S composition is commutative; thus

$$P \otimes R = R \otimes P \quad (2.53)$$

This holds because the S-norm operation in equation (2.51) is commutative ( $aSb = bSa$ ).

#### 2.10.6.4 Associativity

The sup-T composition is associative; thus

$$P \circ (Q \circ R) = (P \circ Q) \circ R \quad (2.54)$$

This may be proved for the sup-min composition as follows. Let the membership functions of  $P$ ,  $Q$ , and  $R$  be  $\mu_P(x, y)$ ,  $\mu_Q(y, z)$  and  $\mu_R(z, w)$ , respectively. This provides sufficient generality for the present proof. Now consider two sides of equation (2.54). The membership functions are:

$$\begin{aligned} LHS &= \sup_y \min[\mu_P(x, y), \sup_z \min(\mu_Q(y, z), \mu_R(z, w))] \\ RHS &= \sup_z \min[\sup_y \min(\mu_P(x, y), \mu_Q(y, z)), \mu_R(z, w)] \end{aligned}$$

If sup and min are taken in orthogonal directions (i.e., different subspaces or with respect to different variables) then these two operations commute. Hence, in the RHS expression, the first min (carried out over  $(z, w)$ ) will commute over the sup carried out over  $y$ . Hence

$$RHS = \sup_z \sup_y \min[\min(\mu_P(x, y), \mu_Q(y, z)), \mu_R(z, w)]$$

Next, since sup commutes with itself and since min of a set of elements can be carried out any two at a time, we have

$$RHS = \sup_y \sup_z \min[(\mu_P(x, y), \min(\mu_Q(y, z), \mu_R(z, w)))]$$

In this expression, the second sup is over  $z$ , and the first min is over  $(x, y)$ . Hence, they commute. We have

$$RHS = \sup_y \min[\mu_P(x, y), \sup_z \min(\mu_Q(y, z), \mu_R(z, w))]$$

This result is identical to the LHS.

Note: A simpler proof can be given for the case of discrete membership functions, in view of the matrix multiplication analogy. Specifically, first we observe that the matrix multiplication is associative. Next we observe that matrix multiplication involves product (analogous to *min*) and addition

(analogous to *max*). Finally, we note that product is associative and so is min; addition is associative and so is max; and product is distributive over addition, and min is distributive over max. This completes the proof.

The inf-S composition is associative; thus

$$P \otimes (Q \otimes R) = (P \otimes Q) \otimes R \quad (2.55)$$

This may be proved for the inf-max composition by following a similar procedure to what was given for the sup-min composition.

#### 2.10.6.5 Distributivity

The sup-T composition is distributive over union ( $\cup$ ). Specifically,

$$(P \cup Q) \circ R = (P \circ R) \cup (Q \circ R) \quad (2.56)$$

In general, the sup-T composition does not distribute over intersection ( $\cap$ ). Specifically,

$$(P \cap Q) \circ R \subset (P \circ R) \cap (Q \circ R) \quad (2.57)$$

Graphical proofs may be provided for these, using special cases.

#### 2.10.6.6 DeMorgan's Laws

Since inf-S composition is the complement of the sup-T composition, DeMorgan's Laws hold; specifically,

$$\overline{P \circ R} = \bar{P} \otimes \bar{R} \quad (2.58a)$$

$$\overline{P \otimes R} = \bar{P} \circ \bar{R} \quad (2.58b)$$

where the over-bar denotes the negation operation for a relation.

#### 2.10.6.7 Inclusion

The inclusion property states that if  $R_1 \subset R_2$  then

$$P \circ R_1 \subset P \circ R_2 \quad (2.59)$$

This property is particularly useful in knowledge-based decision-making with incomplete knowledge or data. Specifically, since  $R_1$  and  $R_2$  may represent two knowledge bases or two sets of data, from (2.59) we conclude that a more complete knowledge base provides a more complete inference, or a more complete set of data provides a more complete inference. Furthermore, (2.59) may be used to show the equivalence between the decisions made by using composition (i.e., CRI) and those made by first determining the decisions corresponding to the individual rules and then aggregating these individual decisions (i.e., individual rule-based inference). This topic will be further discussed in Chapter 3.

### Example 2.26

Suppose that a fuzzy rule base (relation)  $P(x, y)$  is used to make an inference  $B(y)$  from a context  $A(x)$ . Also, suppose that another fuzzy rule base  $Q(y, z)$  is used to make an inference  $C(z)$  from context  $B(y)$ . From this information, derive a suitable rule base  $R(x, z)$  that may be used to make an inference  $C(z)$  from context  $A(x)$ . You must justify your derivation.

The relation from a possible “pre-admission grade” ( $x_i$ ) to “admission to a group of universities” ( $y_j$ ) for a particular student is given by the discrete fuzzy relation:

$$P(x_i, y_j) = \begin{bmatrix} 0.4 & 0.3 & 0.1 \\ 0.5 & 0.4 & 0.2 \\ 0.6 & 0.5 & 0.4 \\ 0.8 & 0.7 & 0.5 \\ 1.0 & 0.8 & 0.6 \end{bmatrix}$$

The relation from a possible “admission to a university” ( $y_j$ ) to possible “completion time” ( $z_k$ ) of degree is given by the discrete relation:

$$Q(y_j, z_k) = \begin{bmatrix} 1.0 & 0.7 & 0.6 & 0.4 \\ 0.9 & 1.0 & 0.5 & 0.3 \\ 0.5 & 1.0 & 0.8 & 0.5 \end{bmatrix}$$

Determine a suitable relation  $R(x_i, z_k)$  that relates possible pre-admission grade ( $x_i$ ) to possible completion time ( $z_k$ ) of degree. Use both *max-min* composition and *max-dot* compositions.

#### **Solution**

$$B(y) = A(x) \circ P(x, y) \quad (\text{i})$$

$$C(z) = B(y) \circ Q(y, z) \quad (\text{ii})$$

Substitute (i) in (ii):

$$C(z) = (A(x) \circ P(x, y)) \circ Q(y, z)$$

which, in view of the associativity property of the composition operation ( $\circ$ ) can be written:

$$\begin{aligned} C(z) &= A(x) \circ [P(x, y) \circ Q(y, z)] \\ &= A(x) \circ R(x, z) \end{aligned}$$

where the equivalent rule base is

$$R(x, z) = P(x, y) \circ Q(y, z)$$

Now, for the given numerical problem:

$$R(x_i, z_k) = \begin{bmatrix} 0.4 & 0.3 & 0.1 \\ 0.5 & 0.4 & 0.2 \\ 0.6 & 0.5 & 0.4 \\ 0.8 & 0.7 & 0.5 \\ 1.0 & 0.8 & 0.6 \end{bmatrix} \circ \begin{bmatrix} 1.0 & 0.7 & 0.6 & 0.4 \\ 0.9 & 1.0 & 0.5 & 0.3 \\ 0.5 & 1.0 & 0.8 & 0.5 \end{bmatrix}$$

Max-min composition:

$$R(x_i, z_k) = \begin{bmatrix} 0.4 & 0.4 & 0.4 & 0.4 \\ 0.5 & 0.5 & 0.5 & 0.4 \\ 0.6 & 0.6 & 0.6 & 0.4 \\ 0.8 & 0.7 & 0.6 & 0.5 \\ 1.0 & 0.8 & 0.6 & 0.5 \end{bmatrix}$$

Max-dot composition:

$$R(x_i, z_k) = \begin{bmatrix} 0.4 & 0.3 & 0.24 & 0.16 \\ 0.5 & 0.4 & 0.3 & 0.2 \\ 0.6 & 0.5 & 0.36 & 0.24 \\ 0.8 & 0.7 & 0.48 & 0.32 \\ 1.0 & 0.8 & 0.6 & 0.4 \end{bmatrix}$$

## 2.10.7 Extension principle

We have noticed that the extension principle applies to decision-making with fuzzy information (fuzzy data) and a crisp knowledge base (crisp relation), while the composition applies to decision-making with fuzzy information (fuzzy data) and a fuzzy knowledge base (fuzzy relation). It follows that the former is a special case of the latter. This may be proved as follows.

Consider a crisp relation  $y = f(x)$ . This may be treated as a fuzzy relation  $R$  with the following membership function:

$$\begin{aligned} \mu_R(x, y) &= 1 && \text{if } y = f(x) \\ &= 0 && \text{otherwise} \end{aligned} \tag{i}$$

Now consider a set of fuzzy data  $P$  with membership function  $\mu_P(x)$ . Then the compositional rule of inference provides an inference with the membership function:

$$\mu_{P \circ R}(y) = \sup_{x \in X} \min(\mu_P(x), \mu_R(x, y)) \tag{ii}$$

Note that in (ii) the *sup* operation is performed over the entire universe X of the variable  $x$ . Instead, let us first carry out the *sup* in the region where the given crisp relation is satisfied, then we carry out the *sup* in the remaining region of the universe, and finally get the *maximum* of the two results. This procedure should give the same result as (ii). Hence

$$\mu_{P \circ R}(y) = \max \left[ \sup_{x=f^{-1}(y)} \min(\mu_P(x), \mu_R(x, y)), \sup_{x \neq f^{-1}(y)} \min(\mu_P(x), \mu_R(x, y)) \right] \quad (\text{iii})$$

Substitute (i) in (iii). We get

$$\mu_{P \circ R}(y) = \max \left[ \sup_{x=f^{-1}(y)} \min(\mu_P(x), 1), \sup_{x \neq f^{-1}(y)} \min(\mu_P(x), 0) \right]$$

Note that the second min gives 0, and hence the second term of the max operation can be dropped. The first min operation simply gives  $\mu_P(x)$ . Hence we get

$$\mu_{P \circ R}(y) = \sup_{x=f^{-1}(y)} \min(\mu_P(x)) \quad (2.60)$$

This result is exactly what the extension principle dictates. This case of single-variable input can be extended to the general case of multivariable input.

## 2.11 Considerations of fuzzy decision-making

A fuzzy knowledge base can represent a complex system not as a set of complex, nonlinear differential equations but as a simple set of input–output relations (or rules). These rules contain fuzzy terms such as small, fast, high, and very, which are common linguistic expressions of knowledge about a practical situation. Designing the rule base is an important step of the development of a fuzzy knowledge-based system. Inferences can be made by matching a set of available information (or observations, or data) with the rule base of the system. This is done by applying the compositional rule of inference. Alternatively, inferences can be made by matching data with individual rules and then combining (aggregating) these individual rule-based inferences (say, using the *max* operation). The latter approach gives identical inferences to what one would get by applying the CRI, albeit more conveniently.

The *antecedent* part of a rule corresponds to the input variables (input space) of the fuzzy decision-making system. Several input variables may be connected by AND connectives; for example, “IF Speed is  $S_i$  AND Distance is  $D_j$ , AND . . .” Here,  $S_i$  and  $D_j$  are fuzzy states of the corresponding fuzzy variables. Different rules in the rule base will involve different fuzzy states of the antecedent variables. Each antecedent variable is assumed to be orthogonal to (independent of) another antecedent variable, and occupies a disjoint subspace of the input space. But there exists some overlap between various fuzzy states of a particular antecedent variable, as provided in a fuzzy context where, in the real world, changes are not abrupt and rather smooth.

The *consequent* part of a rule gives prescribed action (inference, decision). Since, without loss of generality, multiple action variables can be separated into rules having single action variables, it is adequate to only consider rules with single consequent variables.

### 2.11.1 Extensions to fuzzy decision-making

Thus far we have considered fuzzy rules of the form:

$$\text{IF } x \text{ is } A_i \text{ AND IF } y \text{ is } B_i \text{ THEN } z \text{ is } C_i \quad (2.61)$$

where  $A_i$ ,  $B_i$ , and  $C_i$  are fuzzy states governing the  $i$ -th rule of the rule base. In fact this is the Mamdani approach (Mamdani system or Mamdani model) named after the person who pioneered the application of this approach. Here, the knowledge base is represented as fuzzy protocols of the form (2.61) and represented by membership functions for  $A_i$ ,  $B_i$ , and  $C_i$ , and the inference is obtained by applying the compositional rule of inference. The result is a fuzzy membership function, which typically has to be *defuzzified* for use in practical tasks.

Several variations to this conventional method are available. One such version is the *Sugeno model* (or *Takagi–Sugeno–Kang model* or *TSK model*). Here, the knowledge base has fuzzy rules with crisp functions as the consequent, of the form

$$\text{IF } x \text{ is } A_i \text{ AND IF } y \text{ is } B_i \text{ THEN } c_i = f_i(x, y) \quad (2.62)$$

for Rule  $i$ , where  $f_i$  is a crisp function of the condition variables (antecedent)  $x$  and  $y$ . Note that the condition part of this rule is the same as for the Mamdani model (2.61), where  $A_i$  and  $B_i$  are fuzzy sets whose membership functions are functions of  $x$  and  $y$ , respectively. The action part is a crisp function of the condition variables, however. The inference  $\hat{c}(x, y)$  of the fuzzy knowledge-based system is obtained directly as a crisp function of the condition variables  $x$  and  $y$ , as follows.

For Rule  $i$ , a weighting parameter  $w_i(x, y)$  is obtained corresponding to the condition membership functions, as for the Mamdani approach, by using either the “min” operation or the “product” operation. For example, using the “min” operation we form

$$w_i(x, y) = \min[\mu_{A_i}(x), \mu_{B_i}(y)] \quad (2.63)$$

The crisp inference  $\hat{c}(x, y)$  is determined as a weighted average of the individual rule inferences (crisp)  $c_i = f_i(x, y)$  according to

$$\hat{c}(x, y) = \frac{\sum_{i=1}^r w_i c_i}{\sum_{i=1}^r w_i} = \frac{\sum_{i=1}^r w_i(x, y) f_i(x, y)}{\sum_{i=1}^r w_i(x, y)} \quad (2.64)$$

where  $r$  is the total number of rules. For any data  $x$  and  $y$ , the knowledge-based action  $\hat{e}(x, y)$  can be computed from (2.64), without requiring any defuzzification. The Sugeno model is particularly useful when the actions are described analytically through crisp functions, as in conventional crisp control, rather than linguistically. The TSK approach is commonly used in applications of direct control and in simplified fuzzy models. The Mamdani approach, even though popular in low-level direct control, is particularly appropriate for knowledge representation and processing in expert systems and in high-level (hierarchical) control systems.

## 2.12 Summary

Fuzzy logic was first developed by L.A. Zadeh in the mid-1960s for representing some types of “approximate” knowledge that cannot be represented by conventional, crisp methods. In the crisp Boolean logic, truth is represented by the state 1 and falsity by the state 0. Boolean algebra and crisp logic have no provision for approximate reasoning. Fuzzy logic is an extension of crisp bivalent (two-state) logic in the sense that it provides a platform for handling approximate knowledge. Fuzzy logic is based on fuzzy set theory in a similar manner to how crisp bivalent logic is based on crisp set theory. A fuzzy set is represented by a *membership function*. A particular “element” value in the range of definition of the fuzzy set will have a grade of membership, which gives the degree to which the particular element belongs to the set. In this manner, it is possible for an element to belong to the set at some degree and simultaneously not belong to the set at a complementary degree, thereby allowing a non-crisp (or fuzzy) membership. This chapter presented the basic theory of fuzzy logic, which uses the concept of fuzzy sets. Applications of soft computing and particularly fuzzy knowledge-based systems rely on the representation and processing of knowledge using fuzzy logic. In particular, the *compositional rule of inference* is what is applied in decision-making with fuzzy logic. Underlying principles were systematically presented in this chapter. Geometrical illustrations and examples were presented in order to facilitate the interpretation. The application of these concepts in knowledge-based “intelligent” systems and particularly in intelligent control will be treated in future chapters.

### Problems

- Suppose that  $A$  denotes a set,  $X$  denotes the universal set and  $\phi$  denotes the null set.

The *law of contradiction*:

$$A \wedge \bar{A} = \phi$$

and the *law of excluded middle*:

$$A \vee \bar{A} = X$$

- are satisfied by crisp sets and hence agreeable to bivalent logic. Show that these two laws are not satisfied by fuzzy sets and hence violated in fuzzy logic. Give an example to illustrate each of these two cases.
2. Show that for crisp sets, the Bold Intersection (Bounded Max) operation is identical to the conventional intersection (min), and the Bold Union (Bounded Min) operation is identical to the conventional union (max). Also show that this is not the case in general for fuzzy sets. You may use a graphical proof.
  3. Consider the laws of bivalent (crisp) logic. Which of these laws are violated by fuzzy logic? In particular, show that DeMorgan's Laws are satisfied.
  4. Critically investigate why Japanese industry and society accepted and utilized fuzzy logic much sooner and more profitably than western society.
  5. List five consumer appliances that use fuzzy logic. Indicate how fuzzy logic is used in each of these appliances.
  6. List five areas of possible application of fuzzy logic (not necessarily fuzzy logic control). In each area, discuss the appropriateness or inappropriateness of using fuzzy logic, compared to other (conventional) approaches. Also, for each area, describe one practical application and indicate how fuzzy logic is utilized in that application.
  7. Using common knowledge, experience, judgment, and perception, construct and sketch appropriate membership functions for the following sets:
 

(i) Tall men	(v) Hot outside temperature
(ii) Tall women	(vi) Cold outside temperature
(iii) Hot room temperature	(vii) Fast car speed on interstate highway
(iv) Cold room temperature	(viii) Fast car speed in city

 In each case, you must give either the functional relation for the membership function, with appropriate numerical values for their parameters, or numerical data to completely represent the membership function.
  8. Two fuzzy sets  $A$  and  $B$  are represented by the following two membership functions:
 
$$\mu_A(x) = \max\left(0, \frac{x-3}{7}\right) \text{ for } x \leq 10$$

$$= \max\left(0, \frac{17-x}{7}\right) \text{ for } x > 10$$

$$\mu_B(x) = \max\left(0, \frac{x-8}{2}\right) \text{ for } x \leq 10$$

$$= \max\left(0, \frac{12-x}{2}\right) \text{ for } x > 10$$

(a) Sketch these membership functions.  
 (b) What do  $A$  and  $B$  approximately represent?  
 (c) Which one of the two sets is fuzzier?
  9. Consider a fuzzy set  $A$  in the universe  $\mathfrak{R}$  (i.e., the real line) whose membership function is given by

$$\mu_A(x) = \begin{cases} 1 - |x - 2| & \text{for } |x - 2| \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

- (a) Sketch the membership function.  
 (b) What is the support set of  $A$ ?  
 (c) What is the  $\alpha$ -cut of  $A$  for  $\alpha = 0.5$ ?

10. The chronology of *artificial intelligence* may be summarized as follows:

The first technical paper on the subject appeared in 1950. LISP programming language and the first expert system were developed in the 1960s. The first paper on “Fuzzy Sets” was published by Zadeh in 1964. Systems with natural language capabilities appeared in the 1970s. Many hardware tools, development systems, and expert systems were commercially available, and proliferation of fuzzy logic applications, particularly in consumer appliances, took place in the 1980s. Integration of various AI techniques, extensive availability of PC-based systems, and sophisticated user interfaces are noted in the 1990s. Application in decision support systems and other complex industrial systems is continuing in the 2000s.

In your opinion why did it take about two decades for us to accept the versatility to *fuzzy logic* in AI applications?

11. The characteristic function  $\chi_A$  of a crisp set  $A$  is analogous to the membership function of a fuzzy set, and is defined as follows:

$$\begin{aligned}\chi_A(x) &= 1 \quad \text{if } x \in A \\ &= 0 \quad \text{otherwise}\end{aligned}$$

Show that

$$\begin{aligned}\chi_{A'} &= 1 - \chi_A \\ \chi_{A \vee B} &= \max(\chi_A, \chi_B) \\ \chi_{A \wedge B} &= \min(\chi_A, \chi_B) \\ \chi_{A \rightarrow B}(x, y) &= \min[1, \{1 - \chi_A(x) + \chi_B(y)\}]\end{aligned}$$

where  $A$  and  $B$  are defined in the same universe  $X$ , except in the last case (implication) where  $A$  and  $B$  may be defined in two different universes  $X$  and  $Y$ .

What are the implications of these results?

12. Show that the *commutative* and *associative* properties and *boundary conditions* are satisfied for “*min*” and “*max*” norms. Also show that DeMorgan’s Law is satisfied by these two norms. This verifies that “*min*” is a T-norm and “*max*” is an S-norm.
13. For two membership functions  $\mu_A$  and  $\mu_B$  show that

$$\mu_A \cdot \mu_B \leq \min(\mu_A, \mu_B)$$

Does this indicate that the “*max-dot*” inference tends to provide fuzzier inferences (more conservative) than the “*max-min*” inference? Discuss.

14. If  $\mu_A < 0.5$ , show that a set  $A^*$  that is less fuzzy than  $A$  satisfies  $\mu_{A^*} < \mu_A < 0.5$ .

Similarly, if  $\mu_A > 0.5$ , a set  $A^*$  that is less fuzzy than  $A$  satisfies  $\mu_{A^*} > \mu_A > 0.5$ .

15. The degree of containment  $C_{A,B}$  of a fuzzy set  $A$  in another fuzzy set  $B$ , both of which being defined in the same universe, is given by

$$\begin{aligned}C_{A,B} &= 1 \quad \text{if } \mu_A \leq \mu_B \\ &= \mu_B \quad \text{if } \mu_A > \mu_B\end{aligned}$$

What does this parameter represent?

- 16.** Consider the S-norm (or T-conorm) given by  $x + y - xy$  with  $0 < x < 1$  and  $0 < y < 1$ . Show that

$$x + y - xy > \max(x, y)$$

where  $\max(x, y)$  is another S-norm.

- 17.** In the usual notation, the T-norm of two membership functions  $x$  and  $y$  is denoted by  $xTy$  and its complementary (or conjugate) norm, the S-norm, is denoted by  $xSy$ . The two norms are related through the DeMorgan's Law

$$xSy = 1 - (1 - x)T(1 - y)$$

Using this relationship, determine the S-norm corresponding to each of the following two T-norms:

- (i)  $xy$
- (ii)  $\max[0, x + y - 1]$

Clearly indicate all the important steps of your derivations.

In operations with fuzzy sets, what do T-norm and S-norm generally represent?

- 18.** The *transitivity property* of conventional (crisp) sets states that

If  $A \subset B$  and  $B \subset C$  then  $A \subset C$

Is this property satisfied by fuzzy sets? Explain.

- 19.** The *involution property* of fuzzy sets states that

$$\bar{\bar{A}} = A$$

(i.e.,  $\text{NOT}(\text{NOT } A) = A$ ). Is this property satisfied by fuzzy sets? Explain.

- 20.** Consider the fuzzy implication  $A \rightarrow B$  where  $A$  and  $B$  are fuzzy sets. Give several mathematical operations (e.g., min) that are used in the literature for representing fuzzy implication. Discuss the suitability of each operation in practical applications.

- 21.** Suppose that the state of “fast speed” of a machine is denoted by the fuzzy set  $F$  with membership function  $\mu_F(v)$ . Then the state of “very fast speed,” where the *linguistic hedge* “very” has been incorporated, may be represented by  $\mu_F(v - v_0)$  with  $v_0 > 0$ . Also, the state “presumably fast speed,” where the linguistic hedge “presumably” has been incorporated, may be represented by  $\mu_F^2(v)$ .

- (a) Discuss the appropriateness of the use of these membership functions to represent the respective linguistic hedges.
- (b) In particular, if

$$F = \left\{ \frac{0.1}{10}, \frac{0.3}{20}, \frac{0.6}{30}, \frac{0.8}{40}, \frac{1.0}{50}, \frac{0.7}{60}, \frac{0.5}{70}, \frac{0.3}{80}, \frac{0.1}{90} \right\}$$

in the discrete universe  $V = \{0, 10, 20, \dots, 190, 200\}$  rev/s and  $v_0 = 50$  rev/s, determine the membership functions of “very fast speed” and “presumably fast speed.”

- (c) Suppose that power  $p$  is given by the relation (crisp)

$$p = v^2$$

For the given fast speed (fuzzy)  $F$ , determine the corresponding membership function for power. Compare/contrast this with “presumably fast speed.”

- 22.** Sketch the membership function  $\mu_A(x) = e^{-\lambda(x-a)^n}$  for  $\lambda = 2$ ,  $n = 2$ , and  $a = 3$  for the support set  $S = [0, 6]$ . On this sketch separately show the shaded areas that represent the following fuzziness measures:

$$(a) M_1 \int_S f(x)dx \quad \text{where} \quad f(x) = \mu_A(x) \quad \text{for} \quad \mu_A(x) \leq 0.5 \\ = 1 - \mu_A(x) \quad \text{for} \quad \mu_A(x) > 0.5$$

$$(b) M_2 \int_S |\mu_A(x) - \mu_{A_{1/2}}(x)| dx$$

where  $\mu_{A_{1/2}}$  is the  $\alpha$ -cut of  $\mu_A(x)$  for  $\alpha = 1/2$

$$(c) M_3 \int_S |\mu_A(x) - \mu_{\bar{A}}(x)| dx$$

where  $\bar{A}$  is the complement of the fuzzy set  $A$ . Evaluate the values of  $M_1$ ,  $M_2$  and  $M_3$  for the given membership function.

- (i) Establish relationships between  $M_1$ ,  $M_2$  and  $M_3$ .
- (ii) Indicate how these measures can be used to represent the degree of fuzziness of a membership function.
- (iii) Compare your results with the case  $\lambda = 1$ ,  $a = 3$ , and  $n = 2$  for the same support set, by showing the corresponding fuzziness measures on a sketch of the new membership function.

- 23.** The grade of inclusion of fuzzy set  $A$  in fuzzy set  $B$  is given by:

$$g(x) = 1 \quad \text{for} \quad \mu_A(x) \leq \mu_B(x) \\ = \mu_B(x) \quad \text{for} \quad \mu_A(x) > \mu_B(x)$$

Show that  $g(x) = \sup\{c \in [0, 1] | \mu_A(x) T c \leq \mu_B(x)\}$

where the T-norm,  $T$ , may be interpreted as the *min* operation.

- 24.** For a fuzzy set  $A$ , prove the representation theorem:

$$\mu_A(x) = \sup_{\alpha \in [0,1]} [\alpha \mu_{A_\alpha}(x)]$$

where  $\mu_{A_\alpha}$  is the  $\alpha$ -cut of  $\mu_A(x)$ .

- 25.** Show that  $\max[0, x + y - 1]$  is a T-norm. Also, determine the corresponding T-conorm (i.e., S-norm). Hint: Show that the nondecreasing, commutative, and associative properties and the boundary conditions are satisfied.

- 26.** (a) Establish the “isomorphism” between conventional set theory and binary propositional logic by comparing the primary operations of sets and logic.  
 (b) Consider the case of ternary (three-valued) logic where in addition to “True” and “False” with the respective truth values 1 and 0, there is “indeterminacy” with the corresponding truth value  $1/2$ . In this logic, give suitable truth tables for the operations NOT, AND, OR and IF-THEN.

- 27.** (a) Consider the membership function  $\mu_A(x) = e^{-\lambda|x-a|^n}$ , for a fuzzy set  $A$ . Interpret the meaning of the parameters  $a$ ,  $\lambda$  and  $n$ . In particular, discuss how (1) fuzziness and (2) a fuzzy adjective or fuzzy modifier such as “very” or “somewhat” of a fuzzy state may be represented using these parameters.  
 (b) Using the general membership function expression used in part (a), give an analytical representation for temperature inside a living room that has the

three fuzzy states “cold,” “comfortable,” and “hot.” You must give appropriate numerical values for the parameters of the analytical expression.

- 28.** A fuzzy set  $A$  is defined in the universe  $X = [0, 8]$  and its membership function is given by

$$\mu_A(x) = \frac{x+1}{x+2}$$

Determine the fuzzy set  $B$  which is obtained through the crisp relation  $y = x + 1$ . What is the corresponding universe?

- 29.** (a) The extension principle may be considered as a special case of the application of the compositional rule of inference. Describing the extension principle and the compositional rule of inference, justify this statement.  
 (b) A process is represented by the crisp relation

$$z^2 = (x - a)^2 + y^2$$

where information in the  $X \times Y$  domain is mapped to the  $Z$  domain, with  $x, y$ , and  $z$  defined in the universes  $X, Y$ , and  $Z$ , respectively.

Suppose that the information  $S$  in the  $X \times Y$  domain is represented by the fuzzy relation

$$\mu_S(x, y) = \frac{1}{\exp\left[\frac{(x-a)^2}{b} + \frac{(y-c)^2}{d}\right]}$$

What is the membership function  $\mu_C(z)$  of the corresponding “inference”  $C$  in the  $Z$  domain? Assume that  $a, b, c, d > 0$  and  $b > d$ .

Sketch  $\mu_S(x, y)$  and  $\mu_C(z)$ .

- 30.** (a) Define the following terms as applied to fuzzy logic:

- (i) Projection
- (ii) Cylindrical extension
- (iii) Join
- (iv) Composition.

What is the significance of the “composition” operation in fuzzy decision-making?

- (b) Consider two fuzzy relations  $R$  and  $S$ , defined in  $X \times Y$  and  $Y \times Z$ , respectively, as given below.

$$\mu_R(x, y) = 1/\exp\left[\left(\frac{x-a}{b}\right)^2 + \left(\frac{y-c}{d}\right)^2\right]$$

$$\mu_S(y, z) = 1/\exp\left[\left(\frac{y-g}{h}\right)^2 + \left(\frac{z-p}{a}\right)^2\right]$$

- (i) What is the cylindrical extension  $C(R)$  of  $R$  in  $X \times Y \times Z$ ?
- (ii) What is the cylindrical extension  $C(S)$  of  $S$  in  $X \times Y \times Z$ ?
- (iii) What is the join  $J(R, S)$  of  $R$  and  $S$ ?

- (iv) What is the composition  $R \circ S$  of  $R$  and  $S$ ?
- (v) What is the projection of  $R \circ S$  onto  $Z$ ?
- (vi) What is the projection  $P(R)$  of  $R$  onto  $Y$ ?
- (vii) What is the composition of  $P(R)$  and  $S$ ?

Compare this result with what you obtained in part (v) above.

*Note:* You may use the “product” operation instead of the “min” operation to represent the “intersection” of two fuzzy sets.

- 31.** Consider the two fuzzy relations  $R(x, z)$  and  $S(z, y)$ . Their sup-min composition is denoted by  $R \circ S$  where:

$$\mu_{R \circ S}(x, y) = \sup_z \{\min[\mu_R(x, z), \mu_S(z, y)]\}$$

Their inf-max composition is denoted by  $R \otimes S$  where:

$$\mu_{R \otimes S}(x, y) = \inf_z \{\max[\mu_R(x, z), \mu_S(z, y)]\}$$

Carefully check whether the DeMorgan’s Laws:

$$\overline{R \circ S} = \bar{R} \otimes \bar{S}$$

and

$$\overline{R \otimes S} = \bar{R} \circ \bar{S}$$

are satisfied for these compositions. Proofs have to be provided for your conclusions.

- 32.** Consider the binary relations (rule bases)  $R(x, z)$ ,  $P(x, z)$  and the context fuzzy set  $S(z)$ . Check, giving necessary proofs, whether the following relations are satisfied in this case:

- (a)  $(R \cup P) \circ S = (R \circ S) \cup (P \circ S)$
- (b)  $(R \cap P) \circ S = (R \circ S) \cap (P \circ S)$

Indicate practical implications of your results in the context of fuzzy knowledge-based decision-making.

- 33.** Consider the knowledge base represented by the set of fuzzy rules  $Y \rightarrow U$  with

$$Y = \frac{0.2}{y_1} + \frac{0.6}{y_2} + \frac{1.0}{y_3} + \frac{0.6}{y_4} + \frac{0.2}{y_5}$$

and

$$U = \frac{0.3}{u_1} + \frac{1.0}{u_2} + \frac{0.3}{u_3}$$

Determine the membership function matrix  $R$  of this knowledge base.

Suppose that three fuzzy observations  $Y'$  have been made whose membership functions are:

- (a)  $\mu_{Y'}(y_i) = [0.4 \quad 1.0 \quad 0.4 \quad 0 \quad 0]$
- (b)  $\mu_{Y'}(y_i) = [0.4 \quad 0.8 \quad 0.4 \quad 0 \quad 0]$
- (c)  $\mu_{Y'}(y_i) = [0.4 \quad 0.5 \quad 0.4 \quad 0 \quad 0]$

Determine the corresponding fuzzy inference  $U'$  in each case.

34. For fuzzy information  $\mu_X(x)$  and a fuzzy rule base  $\mu_R(x, y)$ , the compositional rule of inference provides the inference

$$X \circ R = \sup_x \min[\mu_X(x), \mu_R(x, y)]$$

Now consider the special case where the rule base is actually a crisp relation  $y = f(x)$ . Specifically,

$$\begin{aligned}\mu_R &= 1 && \text{if } y = f(x) \\ &= 0 && \text{otherwise}\end{aligned}$$

Show that the extension principle is obtained in this case.

## References

1. De Silva, C.W., and Lee, T.H. (1994) “Fuzzy logic in process control,” *Measurements and Control*, vol. 28, no. 3, pp. 114–24, June.
2. De Silva, C.W. (1997) “Intelligent control of robotic systems,” *International Journal of Robotics and Autonomous Systems*, vol. 21, pp. 221–37.
3. De Silva, C.W. (1995) *Intelligent Control: Fuzzy Logic Applications*, CRC Press, Boca Raton, FL.
4. Dubois, D., and Prade, H. (1980) *Fuzzy Sets and Systems*, Academic Press, Orlando.
5. Jain, L.C., and De Silva, C.W. (1999) *Intelligent Adaptive Control: Industrial Applications*, CRC Press, Boca Raton, FL.
6. Klir, G.J., and Folger, T.A. (1988) *Fuzzy Sets, Uncertainty, and Information*, Prentice Hall, Englewood Cliffs, NJ.
7. Zadeh, L.A. (1979) “A theory of approximate reasoning,” *Machine Intelligence*, J.E. Hayes, D. Michie, and L.I. Mikulich (eds), vol. 9, pp. 149–94, Elsevier, New York.



# Fuzzy logic control

## Chapter outline

<b>3.1</b>	<b>Introduction</b>	<b>137</b>
<b>3.2</b>	<b>Background</b>	<b>138</b>
<b>3.3</b>	<b>Basics of fuzzy control</b>	<b>141</b>
<b>3.4</b>	<b>Fuzzy control architectures</b>	<b>162</b>
<b>3.5</b>	<b>Properties of fuzzy control</b>	<b>180</b>
<b>3.6</b>	<b>Robustness and stability</b>	<b>191</b>
<b>3.7</b>	<b>Summary</b>	<b>202</b>
	<b>Problems</b>	<b>202</b>
	<b>References</b>	<b>219</b>

## 3.1 Introduction

Fuzzy logic control or simply “fuzzy control” belongs to the class of “intelligent control,” “knowledge-based control,” or “expert control.” Fuzzy control uses knowledge-based decision-making employing techniques of fuzzy logic, as presented in a previous chapter, in determining the control actions. In this chapter, the basics of analysis and application of fuzzy logic control are presented. The method of generating an aggregate membership function for a fuzzy rule base is studied. The process of making control decisions by applying the *compositional rule of inference* is explained. Typically the monitored process variables are crisp and they have to be *fuzzified* for the purpose of decision-making. Also, the control decisions generated in this manner are fuzzy quantities, and they have to be *defuzzified* prior to using them in a physical action of control. These methods are given. Analytical characteristics and stability considerations of fuzzy control are explored. Hierarchical control corresponds to an architecture that is commonly used in distributed control systems and is particularly relevant when various control functions in the system can be conveniently ranked into different levels. In this chapter, some attempt is made to differentiate between high-level control and low-level direct control in a hierarchical control system. It is hinted that fuzzy logic is more suited for the former. Examples are given to illustrate the various steps of fuzzy logic control.

## 3.2 Background

Automation and autonomous operation of a process call for fast and accurate control schemes. In the past, linear proportional-integral-derivative (PID) control has been successfully utilized for low-level operations in process control, but mainly in slow processes whose response approximated that of linear models. At high speeds, and in the presence of dynamic coupling, nonlinearities, and parameter variations, more sophisticated control techniques would be needed, particularly for high-order plants. Many control algorithms have been developed to address a variety of problems in complex plants of this type. So-called “modern” control methods include linear quadratic Gaussian (LQG) control, modal control, linearizing feedback control (or feedback linearization technique or FLT), various forms of adaptive control (e.g., model-referenced adaptive control), H-infinity control, and sliding mode control (or variable structure control). There are many reasons for the practical deficiencies of conventional, crisp algorithmic control. Conventional control is typically based on “modeling” the plant that is to be controlled. If the model is not accurate the performance of the controller may not be acceptable. Furthermore, these conventional controllers rely on a complete set of data, including sensory information and parameter values, to produce control actions, and indeed, program instructions and data are conventionally combined into the same memory of the controller. If the model parameters and other necessary data are not completely known, say unexpectedly as a result of sensing problems, then appropriate estimates have to be made. Furthermore, if the available information is fuzzy, qualitative, or vague, these “crisp” controllers that are based on such *incomplete information* will not usually provide satisfactory results. The environment with which the plant interacts may not be completely predictable either, and it is normally difficult for a crisp control algorithm to accurately respond to a condition that it did not anticipate and that it could not “understand.” Also, some conventional techniques of control assume that the plant is linear and time-invariant, an assumption that hardly holds in a large majority of practical problems. Conventional control techniques that are based on “crisp” algorithms or mathematical formulae can fail when the plant to be controlled is very complex, highly nonlinear, incompletely known, or contains large process delays. Often, a control strategy that is based on “soft knowledge” such as human “experience” in operating the plant, heuristics, common sense, or expert opinion can provide the remedy to these problems. An intelligent controller may be interpreted as a computer-based controller that can somewhat “emulate” the reasoning procedures of a human expert in the specific subject area (control in the present context), to generate the necessary control actions. Here, techniques from the field of *artificial intelligence (AI)* are used for the purpose of acquiring and representing knowledge and for generating control decisions through an appropriate reasoning mechanism. With steady advances in the field of AI, especially pertaining to the development of practical *expert systems* or *knowledge systems*, there has been a considerable interest in using AI techniques for controlling complex processes.

Industrial processes are generally ill-defined, and explicitly modeling them would be a considerably difficult task. In some situations, even experimental modeling (or model identification) is not feasible because the required outputs of the system are not measurable. Even when modeling is possible, the models could be so complex that accurate control algorithms based on these models could seriously reduce the control bandwidth, thereby slowing the process and introducing unacceptable time lags. This would in turn lead to system instabilities and response errors. Modern industrial plants and technological products are often required to perform complex tasks with high accuracy, under ill-defined conditions. Conventional control techniques may not be quite effective in these systems. “Soft” intelligent control in general is more appropriate in some applications of process control, for example when the process is complex and incompletely known, and when the control procedure cannot be expressed as a crisp algorithm. Most older types of industrial processes are manually supervised and controlled.

It is common knowledge in process control practice that when an experienced control engineer is included as an advisor to a control loop, significant improvements in the system performance are usually possible. Improvements in performance are gained through high-level control actions (tuning actions in particular) of an experienced and skilled operator. Generally, adjusting the operating variables and control settings, tuning, and other control actions that are carried out by a human operator are represented and implemented, not through hard (crisp) algorithms, but rather qualitatively using linguistic rules based on common sense, heuristics, knowledge, and experience. For example, an expert might teach a process operator the necessary control actions using a set of protocols containing linguistic fuzzy terms such as “fast,” “small,” and “accurate.” A practical difficulty invariably arises because except in very low bandwidth processes, human actions are not fast enough for this approach to be feasible. Furthermore, it is not economical to dedicate a human expert to every process, as human experts are scarce and costly. Much knowledge might be available in many forms such as operator experience, manuals, reports, and charts, for these processes. This knowledge may be expressed as a set of protocols, which could form the knowledge base for monitoring and controlling the plant. This knowledge base would represent an “implicit” model of the system. Knowledge-based control (or “intelligent control”) that can capture human experience and other forms of “soft” knowledge, and use them in a reasoning framework, would be quite desirable for such processes, and provides a tremendous potential.

The term intelligent control may be loosely used to denote a control technique that can be carried out using the “intelligence” of a human who is knowledgeable in the particular domain of control. In this definition, constraints pertaining to limitations of sensory and actuation capabilities and information processing speeds of humans are not considered. It follows that if a human in the control loop can properly control a plant, then that system would be a good candidate for intelligent control. In an ideal sense, an intelligent controller is expected to mimic the behavior of a human controller. Humans are “soft” or “flexible” (rather than “hard”) systems. They can

adapt to unfamiliar situations, and they are able to gather information efficiently. This information need not be complete or precise, and might be general, qualitative, or fuzzy. This does not cause difficulty, for humans can perceive, reason, infer, and deduce new information. They can learn, gain new knowledge, and improve their performance through experience. A faithful duplication of these “intelligent” characteristics of a human, in an intelligent controller, is hardly feasible. In the present, early stages of development of the field, it is adequate to concentrate on special-purpose controllers which cater to a limited class of problems, rather than attempting to develop a general theory for intelligent control. Knowledge-based control depends on efficient ways of representing and processing the control knowledge. Specifically, a knowledge base has to be developed and a technique of reasoning and making “inferences” has to be available. Knowledge-based intelligent control relies on knowledge that is gained by intelligently observing, studying, or understanding the behavior of a plant, rather than explicitly modeling the plant, to arrive at the control action. In this context, it also heavily relies on the knowledge of experts in the domain, and also on various forms of general knowledge. Modeling of the plant is implicit here. Information abstraction and knowledge-based decision-making that incorporates abstracted information are considered important in intelligent control. Unlike conventional control, intelligent control techniques possess capabilities of effectively dealing with incomplete information concerning the plant and its environment, and unexpected or unfamiliar conditions.

In fuzzy logic control (or fuzzy control) the task of generating control decisions is computer automated, thereby alleviating the problems of control speed and facilitating simultaneous monitoring and control of many variables. There, linguistic descriptions of human expertise in controlling a process are represented as fuzzy rules or relations, and this knowledge base is used, in conjunction with some knowledge of the state of the process (e.g., measured response variables), by an inference mechanism to determine the necessary control actions at a sufficiently fast rate. Fuzzy logic control does not require a conventional model of the process, whereas most conventional control techniques (i.e., model-based control), for example, require either an analytical model or an experimental model. In view of this, fuzzy logic control is particularly suitable for complex and ill-defined processes in which analytical modeling is difficult due to the fact that the process is not completely known, and experimental model identification is not feasible because the required inputs and outputs of the process may not be measurable. Being a knowledge-based control approach, fuzzy logic control has advantages when considerable experience is available regarding the behavior of the process and when this knowledge can be expressed as a set of rules containing fuzzy quantities. Learning and self-improvement can be conveniently incorporated into fuzzy logic control, through, for example, techniques of rule evaluation and modification. The starting point of conventional fuzzy control is the development of a rule base using linguistic descriptions of control protocols, say, of a human expert. This step is analogous to the development of a “hard” (crisp) control algorithm and the identification of parameter values

for the algorithm in a conventional control approach. The rule base consists of a set of IF-THEN rules relating the fuzzy quantities that represent process response (outputs) and control inputs. During the control action, process measurements (context) are matched with rules in the rule base, using the compositional rule of inference, to generate fuzzy control inferences. This inference procedure is clearly analogous to feedback control in a conventional, crisp control scheme. Even though it is incorrect to claim that fuzzy control is always better than the conventional types of “crisp” (or “hard”) control, it has several advantages. “Soft” intelligent control in general, and fuzzy logic control in particular, are more appropriate in some applications of process control, for example when the process is incompletely known, when the control procedure cannot be expressed as a crisp algorithm, and when human experience in controlling the process is available as a set of linguistic statements or rules containing fuzzy terms.

Fuzzy logic control (FLC) has been applied in a number of processes ranging from ship autopilots, subway cars, helicopters, and pilot-scale steam engines to cement kilns, robotic manipulators, fish processing machines, and household appliances. One obvious drawback in many such applications is that fuzzy control laws are implemented at the lowest level of the control system, within a direct-digital-control (DDC) loop, generating control signals for process actuation directly through fuzzy inference. In high-bandwidth processes this form of fuzzy control implementation would require very fast and accurate control in the presence of strong nonlinearities and dynamic coupling. Another drawback of this direct implementation of FLC is that the control signals are derived from inferences that are fuzzy, thereby directly introducing errors into the necessary crisp control signals. A third argument against the conventional, low-level implementation of fuzzy control is that in a high-speed process, human experience is gained not through manual, on line generation of control signals in response to process output, but typically through performing parameter adjustments and tuning (i.e., manual adaptive control) operations. Hence, it can be argued that, in the case of low-level direct fuzzy control, the protocols for generating control signals are established not through direct experience but by some form of correlation. Of course, it is possible to manually examine input-output data recorded from a high-speed process. However, in such off line studies, on line human interaction is not involved, and again the experience gained would be indirect and somewhat artificial. It follows that it is more appropriate to use fuzzy logic for tuning a controller and for similar high-level tasks than for direct control.

### 3.3 Basics of fuzzy control

Fuzzy control uses the principles of fuzzy logic-based decision-making to arrive at the control actions. The decision-making approach is typically based on the *compositional rule of inference* (CRI), as presented in a previous chapter. In essence, some information (e.g., output measurements) from the system to be controlled is matched with a knowledge base of control for the

particular system, using CRI. A fuzzy rule in the knowledge base of control is generally a “linguistic relation” of the form

$$\text{IF } A_i \text{ THEN IF } B_i \text{ THEN } C_i \quad (3.1)$$

where  $A_i$  and  $B_i$  are fuzzy quantities representing process measurements (e.g., process error and change in error) and  $C_i$  is a fuzzy quantity representing a control signal (e.g., change in process input). What we have is a rule base with a set of ( $n$ ) rules:

$$\text{Rule 1: } A_1 \text{ and } B_1 \Rightarrow C_1$$

$$\text{Rule 2: } A_2 \text{ and } B_2 \Rightarrow C_2$$

...   ...   ...   ...

$$\text{Rule } n: A_n \text{ and } B_n \Rightarrow C_n$$

Because these fuzzy sets are related through IF-THEN implications and because an implication operation for two fuzzy sets can be interpreted as a “minimum operation” on the corresponding membership functions, as discussed in a previous chapter, the membership function of this fuzzy relation may be expressed as

$$\mu_{R_i}(a, b, c) = \min[\mu_{A_i}(a), \mu_{B_i}(b), \mu_{C_i}(c)] \quad (3.2)$$

The individual rules in the rule base are joined through ELSE connectives, which are OR connectives (“unions” of membership functions). Hence, the overall membership function for the complete rule base (relation  $R$ ) is obtained using the “maximum” operation on the membership functions of the individual rules; thus

$$\mu_R(a, b, c) = \max_i \mu_{R_i}(a, b, c) = \max_i \min[\mu_{A_i}(a), \mu_{B_i}(b), \mu_{C_i}(c)] \quad (3.3)$$

In this manner the membership function of the entire rule base can be determined (or “identified” in the terminology of conventional control) using the membership functions of the response variables and control inputs. Note that a fuzzy knowledge base is a multivariable function – a multidimensional array (a three-variable function or a dimensional array in the case of equation (3.3)) of membership function values. This array corresponds to a fuzzy control algorithm in the sense of conventional control. The control rule base may represent linguistic expressions of experience, expertise, or knowledge of the domain experts (control engineers, skilled operators, etc.). Alternatively, a control engineer may instruct an operator (or a control system) to carry out various process tasks in the usual manner; monitor and analyze the resulting data; and learn appropriate rules of control, say by using neural networks.

Once a fuzzy control knowledge base of the form given by equation (3.3) is obtained, we need a procedure to infer control actions using process

measurements, during control. Specifically, suppose that fuzzy process measurements  $A'$  and  $B'$  are available. The corresponding control inference  $C'$  is obtained using the compositional rule of inference (i.e., inference using the composition relation). As discussed in a previous chapter, the applicable relation is

$$\mu_{C'}(c) = \sup_{a,b} \min[\mu_{A'}(a), \mu_{B'}(b), \mu_R(a, b, c)] \quad (3.4)$$

Note that in fuzzy inference, the data fuzzy sets  $A'$  and  $B'$  are jointly matched with the knowledge-base fuzzy relation  $R$ . This is a “join” operation, which corresponds to an AND operation (an “intersection” of fuzzy sets), and hence the *min* operation applies for the membership functions. For a given value of control action  $c$ , the resulting fuzzy sets are then mapped (projected) from a three-dimensional space  $X \times Y \times Z$  of knowledge onto a one-dimensional space  $Z$  of control actions. This mapping corresponds to a set of OR connectives, and hence the *sup* operation applies to the membership function values, as expressed in equation (3.4).

Actual process measurements are crisp. Hence, they have to be *fuzzified* in order to apply the compositional rule of inference. This is conveniently done by reading the grade values of the membership functions of the measurement at the specific measurement values. Typically, the control action must be a crisp value as well. Hence, each control inference  $C'$  must be *defuzzified* so that it can be used to control the process. Several methods are available to accomplish defuzzification. In the *mean of maxima* method the control element corresponding to the maximum grade of membership is used as the control action. If there is more than one element with a maximum (peak) membership value, the mean of these values is used. In the *center of gravity* (or *centroid*) method the centroid of the membership function of control decision is used as the value of crisp control action. This weighted control action is known to provide a somewhat sluggish, yet more robust control. Schematic representation of a fuzzy logic controller of the type described here is shown in Figure 3.1. In this diagram, application of the compositional rule of inference has been interpreted as a rule-matching procedure.

There are several practical considerations of fuzzy control that were not addressed in the above discussion. Because representation of the rule base  $R$  by an analytical function can be somewhat unrealistic and infeasible in general, it is customary to assume that the fuzzy sets involved in  $R$  have discrete and finite universes (or at least discrete and finite support sets). As a result, process response measurements must be quantized. Hence, at the outset, a decision must be made as to the element resolution (quantization error) of each universe. This resolution governs the cardinality of the sets and in turn the size of the multidimensional membership function array of a fuzzy rule base. It follows that the required computational effort, memory and storage requirements, and accuracy are directly affected by the quantization resolution.

Because process measurements are crisp, one method of reducing the real-time computational overhead is to precompute a decision table relating

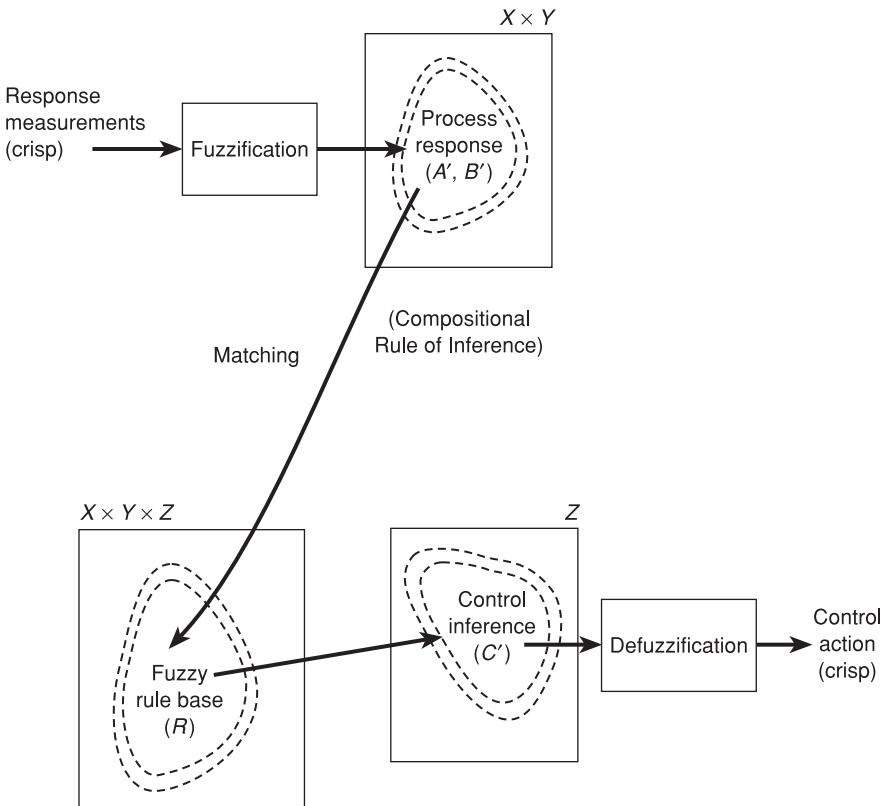


Figure 3.1: A fuzzy logic controller (FLC)

quantized measurements to crisp control actions. The main disadvantage of this approach is that it does not allow for convenient modifications (e.g., rule changes and quantization resolution adjustments) during operation. Another practical consideration is the selection of a proper sampling period in view of the fact that process responses are generally analog signals. Factors such as process characteristics, required control bandwidth, and the processing time needed for one control cycle must be taken into account in choosing a sampling period. Scaling or gain selection for various signals in a fuzzy logic control system is another important consideration. For reasons of processing efficiency, it is customary to scale the process variables and control signals in a fuzzy control algorithm. Furthermore, adjustable gains can be cascaded with these system variables so that they may serve as tuning parameters for the controller. A proper tuning algorithm would be needed, however. A related consideration is real-time or on line modification of a fuzzy rule base. Specifically, rules may be added, deleted, or modified on the basis of some scheme of *learning and self-organization*. For example, using a model for the process and making assumptions such as input-output monotonicity, it is

possible during control to trace and tag the rules in the rule base that need attention. The control-decision table can be modified accordingly.

### 3.3.1 Steps of fuzzy logic control

The main steps of fuzzy logic control, as described above, can be summarized in the following manner. A fuzzy knowledge base must be developed first (off line), according to the following four steps:

- (1) Develop a set of linguistic control rules (protocols) that contain fuzzy variables as conditions (process outputs) and actions (control inputs to the process).
- (2) Obtain a set of membership functions for process output variables and control input variables.
- (3) Using the “fuzzy AND” operation (typically, *min*) and the “fuzzy implication” operation (typically, *min*) on each rule in Step 1, and using Step 2, obtain the multivariable rule base function  $R_i$  (a multidimensional array of membership values in the discrete case) for that rule.
- (4) Combine (aggregate) the relations  $R_i$  using the fuzzy connectives ELSE (fuzzy OR; typically, *max*) to obtain the overall fuzzy rule base (relation)  $R$  (a multidimensional array in the discrete case).

Then, control actions may be determined in real time as follows:

- (1) Fuzzify the measured (crisp) process variables (for example, a fuzzy singleton may be obtained by reading the grade value of the corresponding membership function at the measured value of the variable).
- (2) Match the fuzzy measurements obtained in Step 1 with the membership function (array in the discrete case) of the fuzzy rule base (obtained in previous Step 4), using the compositional rule of inference.
- (3) Defuzzify the control inference obtained in Step 2 (for example, the mean of maxima method or the centroid method may be used here).

These steps reflect the formal procedure in FLC. There are several variations. For example, a much faster approach would be to develop a crisp decision table by combining the four steps of fuzzy knowledge base development and the first two steps of control, and using this table in a table look-up mode to determine a crisp control action during operation. Also, hardware fuzzy processors (*fuzzy chips*) may be used to carry out the fuzzy inference at high speed. The rules, membership functions, and measured context data are generated as usual, through the use of a control “host” computer. The fuzzy processor is located in the same computer, which has appropriate interface (input/output) hardware and driver software. Regardless of all these, it is more convenient to apply the inference mechanism separately to each rule and then combine the result instead of applying it to the entire rule base using the compositional rule of inference. This aspect is discussed next.

### 3.3.2 Composition using individual rules

The compositional rule of inference as represented by equation (3.4) assumes that the system measurements used are themselves fuzzy, represented by the fuzzy labels (fuzzy sets)  $A'$  and  $B'$ . This is true for “human-like,” high-level sensors in which the sensor itself is treated as a fuzzy system. This is also true if the information from low-level crisp sensors is preprocessed by intelligent preprocessors or intelligent filters, to generate fuzzy information for subsequent application of the compositional rule of inference as in equation (3.4).

More common, at least in low-level direct control, is sensory information that is crisp. In fact, such crisp information may be used in equation (3.4) without intelligent preprocessing. In this case, the application of the compositional rule of inference becomes quite simple. To derive the result corresponding to equation (3.4) for this case of directly using crisp measurements in fuzzy inference, we proceed as described below.

Suppose that the rule base  $R$  is given by

$$\text{Else } (A_i, B_j) \rightarrow C_k \quad (3.5)$$

in which the number of possible combinations of fuzzy states  $(i, j)$  of the condition variables (fuzzy sets)  $A$  and  $B$  determine the maximum number of rules in the rule base. The fuzzy state  $k$  of the action variable (fuzzy set)  $C$  is determined by the particular rule that is considered, and hence is dependent on  $(i, j)$ . The corresponding membership function of the rule base is given by (3.3). Now, define a *delta function* as

$$\begin{aligned} \delta(p - p_0) &= 1 \quad \text{when } p = p_0 \\ &= 0 \quad \text{elsewhere} \end{aligned} \quad (3.6)$$

Suppose that the crisp measurements  $(a_0, b_0)$  are obtained from the process outputs  $(a, b)$ . These measurements are “fuzzified” as the fuzzy singletons  $A'$  and  $B'$  with the membership functions:

$$\mu_{A'}(a) = \mu_A(a) \delta(a - a_0); \quad \mu_{B'}(b) = \mu_B(b) \delta(b - b_0) \quad (3.7)$$

Now, since the fuzzy variable  $A$  has the states  $A_1, A_2, \dots$  and the fuzzy variable  $B$  has the states  $B_1, B_2, \dots$ , we have:  $A = A_1 \text{ OR } A_2 \text{ OR } \dots$ , and similarly,  $B = B_1 \text{ OR } B_2 \text{ OR } \dots$ , which gives

$$\begin{aligned} \mu_A(a) &= \max_j \mu_{A_j}(a) \\ \mu_B(b) &= \max_k \mu_{B_k}(b) \end{aligned} \quad (3.8)$$

The corresponding control decision  $C'$  is determined by applying the compositional rule of inference (3.4). Specifically, by substituting equations (3.7), (3.8), and (3.3) into (3.4) we get

$$\begin{aligned}
 \mu_{C'}(c) &= \sup_{a,b} \min [\mu_{A'}(a), \mu_{B'}(b), \mu_R(a, b, c)] \\
 &= \sup_{a,b} \min [\max_j \mu_{A_j}(a) \delta(a - a_0), \max_k \mu_{B_k}(b) \delta(b - b_0), \\
 &\quad \max_i \min \{\mu_{A_i}(a), \mu_{B_i}(b), \mu_{C_i}(c)\}]
 \end{aligned} \tag{3.9}$$

Now when  $a \neq a_0$  the first term on the RHS of equation (3.9) vanishes, and similarly when  $b \neq b_0$  the second term vanishes. Then, in performing the “sup” operation with respect to  $a$  and  $b$ , only the values at  $a = a_0$  and  $b = b_0$  on the RHS of (3.9) will remain. Hence, equation (3.9) becomes

$$\mu_{C'}(c) = \min [\max_j \mu_{A_j}(a_0), \max_k \mu_{B_k}(b_0), \max_i \min \{\mu_{A_i}(a_0), \mu_{B_i}(b_0), \mu_{C_i}(c)\}] \tag{3.9}^*$$

But it is clear that  $\max_j p_j \geq \max_i \min(p_i, q_i)$  for any  $p$  and  $q$ . Hence the first two terms on the RHS of (3.9)\* can never be smaller than the third term, and since the *min* of the three terms is taken, the first two terms can be dropped. Accordingly, (3.9)\* becomes

$$\mu_{C'}(c) = \max_i \min [\mu_{A_i}(a_0), \mu_{B_i}(b_0), \mu_{C_i}(c)] \tag{3.10}$$

The result (3.10) is the most common form of the compositional rule of inference that is employed in decision-making for fuzzy logic control. Specifically, in using (3.10) note that each individual rule  $i$  is considered separately, and the composition is applied in conjunction with the given measurement, to obtain the corresponding inference. Then these individual rule-based inferences are superimposed (i.e., combined through a “max” operation) to provide the overall inference.

It is clear from the derivation of (3.10) that at least for the case of crisp measurements, the individual rule-based inference is equivalent to the original method of aggregate inference, which applies the composition to the complete rule base in a single step. This result is generally true, and may be conveniently proved using the fact that the sup-t composition is distributive over union ( $\cup$ ), as discussed in a previous chapter. Specifically, consider two rules  $R_1$  and  $R_2$ , forming the rule base  $R$ . Clearly,  $R = R_1 \cup R_2$ . Then, we may write the distributivity of composition ( $\circ$ ) over union ( $\cup$ ) as:

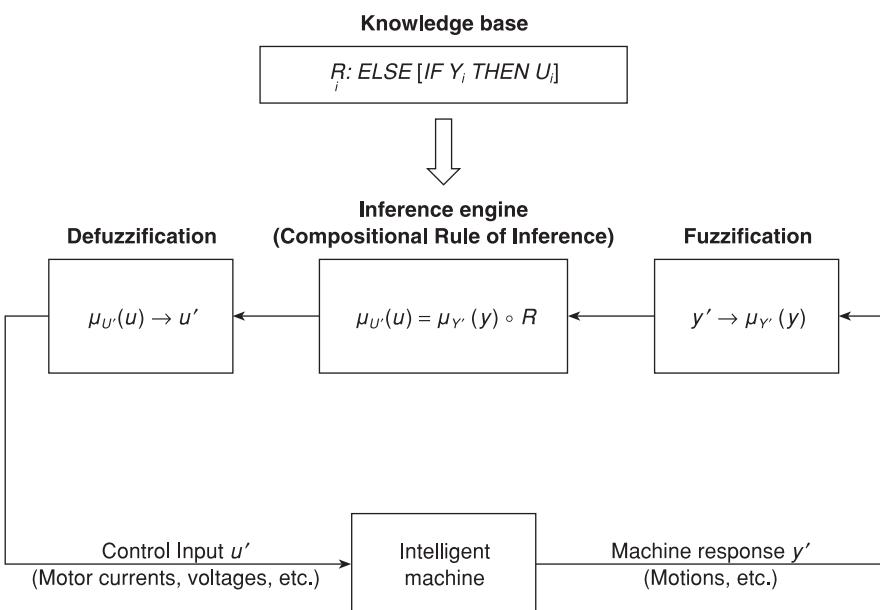
$$D \circ (R_1 \cup R_2) = (D \circ R_1) \cup (D \circ R_2) \tag{3.11}$$

where  $D$  may represent fuzzy data. The property (3.11) states that applying composition to the overall rule base  $R$  is identical to applying composition to the two individual rules and then taking the union of the individual results. Of course, it is straightforward to extend this idea to more than two rules. Hence, the property (3.11) shows the equivalence between decision-making by using composition (i.e., CRI) and that performed by first determining the decisions corresponding to the individual rules and then aggregating these individual decisions (i.e., individual rule-based inference).

In summary, the steps of applying equation (3.10) for individual rule-based decision-making are as follows:

- (1) Consider the first rule,  $i = 1$ . Initiate  $\mu_{C_0}(c) = 0$ .
- (2) For this rule (Rule  $i$ ) read the membership grades at points  $a = a_0$  and  $b = b_0$  in the process output membership functions  $\mu_{A_i}(a)$  and  $\mu_{B_i}(b)$ .
- (3) Take the minimum of the two values of Step 2. Clip the control action membership function  $\mu_{C_i}(c)$  by this value.
- (4) Set  $\mu_{C_i}(c) = \max[\mu_{C_i}(c), \mu_{C_{i-1}}(c)]$
- (5) If  $i = \text{total number of rules}$ , set  $\mu_C(c) = \mu_{C_i}(c)$  and stop. Otherwise, set  $i = i + 1$  and go to Step 2.

It should be clear that in making control inferences, the individual rule-based inference given by equation (3.10) is computationally far more attractive than the aggregate rule-based composition given by equation (3.4). In particular, in equation (3.10) each rule is considered separately and only in terms of the membership grades of the constituent condition variables at the measurement point. Equation (3.4) assumes that the overall multidimensional membership function of the entire rule base ( $R$ ) is available, which involves a large number of *min* and *max* operations. Then the compositional rule of inference is applied in a single step to this membership function, which requires another set of *min* and *sup (max)* operations. The foregoing discussion used just two condition variables for a fuzzy rule. Clearly the ideas can be extended to the case of more than two condition variables.



**Figure 3.2:** Structure of a direct fuzzy controller

Fuzzy logic is commonly used in direct control of processes and machinery. In this case the inferences of a fuzzy decision-making system form the control inputs to the process. These inferences are arrived by using the process responses as the inputs (context data) to the fuzzy system. The structure of a direct fuzzy controller is shown in Figure 3.2. Here,  $y$  represents the process output,  $u$  represents the control input, and  $R$  is the relation, which represents the fuzzy control knowledge base.

### Example 3.1

Consider the room comfort control system schematically shown in Figure 3.3. The temperature ( $T$ ) and humidity ( $H$ ) are the process variables that are measured. These sensor signals are provided to the fuzzy logic controller, which determines the cooling rate ( $C$ ) that should be generated by the air conditioning unit. The objective is to maintain a particular comfort level inside the room.

A simplified fuzzy rule base of the comfort controller is graphically presented in Figure 3.4. The temperature level can assume one of two fuzzy states ( $HG$ ,  $LW$ ), which denote high and low, respectively, with the corresponding membership functions. Similarly, the humidity level can assume two other fuzzy states ( $HG$ ,  $LW$ ) with associated membership functions. Note that the membership functions of  $T$  are quite different from those of  $H$ , even though the same nomenclature is used. There are four rules, as given in Figure 3.4. The rule base is:

Rule 1:	If $T$ is $HG$ and $H$ is $HG$ then $C$ is $PH$
Rule 2:	else if $T$ is $HG$ and $H$ is $LW$ then $C$ is $PL$
Rule 3:	else if $T$ is $LW$ and $H$ is $HG$ then $C$ is $NL$
Rule 4:	else if $T$ is $LW$ and $H$ is $LW$ then $C$ is $NH$
end if	

The nomenclature used for the fuzzy states is as follows:

<i>Temperature (T)</i>	<i>Humidity (H)</i>	<i>Change in cooling rate (C)</i>
$HG = \text{High}$	$HG = \text{High}$	$PH = \text{Positive high}$
$LW = \text{Low}$	$LW = \text{Low}$	$PL = \text{Positive low}$ $NH = \text{Negative high}$ $NL = \text{Negative low}$

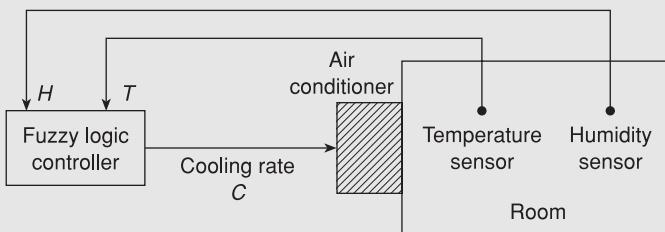


Figure 3.3: Comfort control system of a room

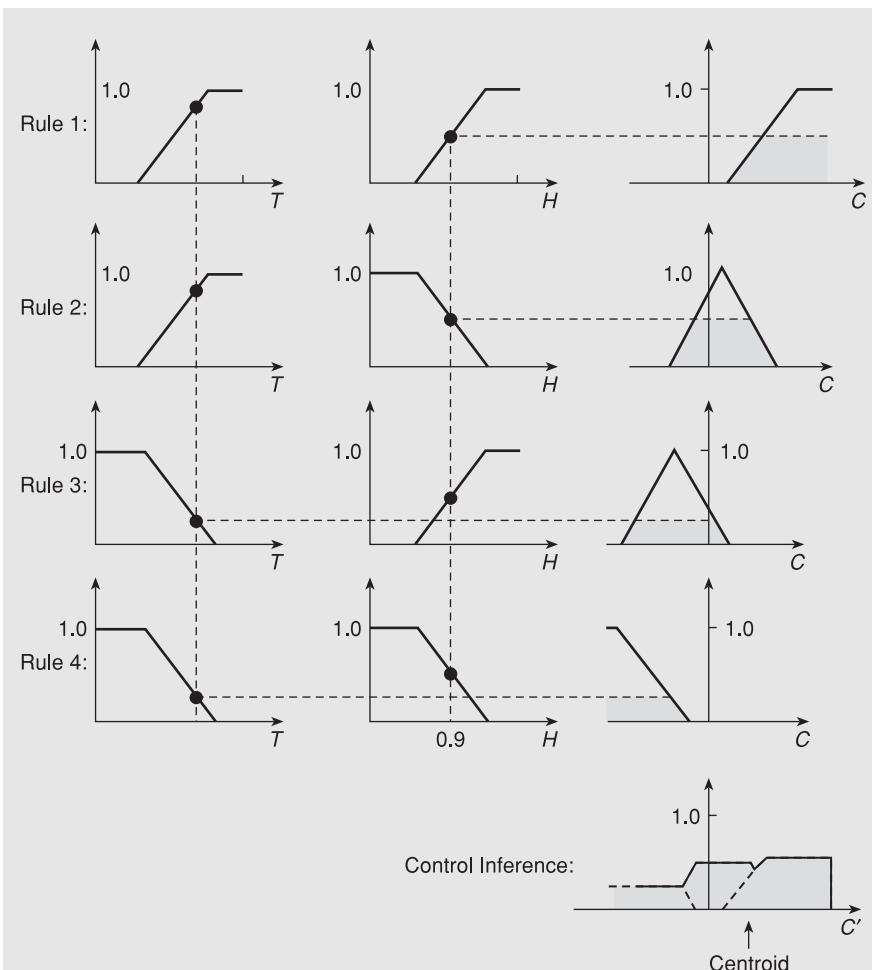


Figure 3.4: The fuzzy knowledge base of the comfort controller

Application of the compositional rule of inference is done here by using individual rule-based composition, as given by equation (3.10). For example, suppose that the room temperature is  $30^\circ\text{C}$  and the relative humidity is 0.9. Lines are drawn at these points, as shown in Figure 3.4, to determine the corresponding membership grades for the fuzzy states in the four rules. In each rule the lower value of the two grades of process response variables is then used to clip (or modulate) the corresponding membership function of  $C$  (a *min* operation). The resulting “clipped” membership functions of  $C$  for all four rules are superimposed (a *max* operation) to obtain the control inference  $C'$  as shown. This result is a fuzzy set, and it must be defuzzified to obtain a crisp control action  $\hat{c}$  for changing the cooling rate. The centroid method may be used, as explained in the next section.

### 3.3.3 Defuzzification

The decision (control action) of a fuzzy logic controller is a fuzzy value and is represented by a membership function. Because low-level control actions are typically crisp, the control inference must be *defuzzified* for physical purposes such as actuation. Several methods are available for defuzzification of a fuzzy control inference.

#### 3.3.3.1 Centroid method

Suppose that the membership function of a control inference is  $\mu_C(c)$ , and its *support set* (the “crisp” set for which the membership grade is greater than zero) is given by (see definition in the chapter on fuzzy logic):  $S = \{c | \mu_C(c) > 0\}$ . Then, the centroid method of defuzzification is expressed as

$$\hat{c} = \frac{\int_{c \in S} c \mu_C(c) dc}{\int_{c \in S} \mu_C(c) dc} \quad (3.12)$$

where  $\hat{c}$  is the defuzzified control action, which is given by the centroid (or *center of gravity*) of the membership function of control inference. Note that equation (3.12) corresponds to the case in which the membership function is continuous. The discrete case is given by

$$\hat{c} = \frac{\sum_{c_i \in S} c_i \mu_C(c_i)}{\sum_{c_i \in S} \mu_C(c_i)} \quad (3.13)$$

#### 3.3.3.2 Mean of maxima method

In the mean of maxima method, if the membership function of the control inference is *unimodal* (i.e., it has just one peak point), the control value at the peak membership grade is chosen as the defuzzified control action. Specifically,

$$\hat{c} = c_{\max} \quad \text{such that} \quad \mu_C(c_{\max}) = \max_{c \in S} \mu_C(c) \quad (3.14)$$

The result for the discrete case follows from this relation.

If the control membership function is *multi-modal* (i.e., has more than one peak), the mean (average) of the control values at these peak points, weighted by the corresponding membership grades, is used as the defuzzified value. Hence, if we have

$$c_i \quad \text{such that} \quad \mu_C(c_i) \Delta = \mu_i = \max_{c \in S} \mu_C(c), \quad i = 1, 2, 3, \dots, p$$

then

$$\hat{c} = \frac{\sum_{i=1}^p \mu_i c_i}{\sum_{i=1}^p \mu_i} \quad (3.15)$$

Here,  $p$  is the total number of modes (peaks) in the control inference membership function.

### 3.3.3.3 Threshold methods

Sometimes, when computing a defuzzified value, it may be desirable to leave out the boundaries of the control inference membership function. In this manner, only the main core of the control inference is used, not excessively diluting or desensitizing the defuzzified value. The corresponding procedures of defuzzification are known as threshold methods. In this case the formulae remain the same as given before, except that the set over which the integration, summation, or peak search is carried out for defuzzification is not the entire support set but an  $\alpha$ -cut (see chapter on fuzzy logic) of the control inference set. Specifically, we select

$$S_\alpha = \{c | \mu_C(c) \geq \alpha\} \quad (3.16)$$

in which  $\alpha$  is the threshold value. Then any one of equations (3.12) through (3.15) may be used, with the support set  $S$  replaced by its  $\alpha$ -cut  $S_\alpha$ . For the centroid method, thresholded defuzzification is illustrated in Figure 3.5.

### 3.3.3.4 Comparison of the defuzzification methods

The *centroid method* uses the entire membership function of a control inference. Thus, the defuzzified value depends on both size and shape of the

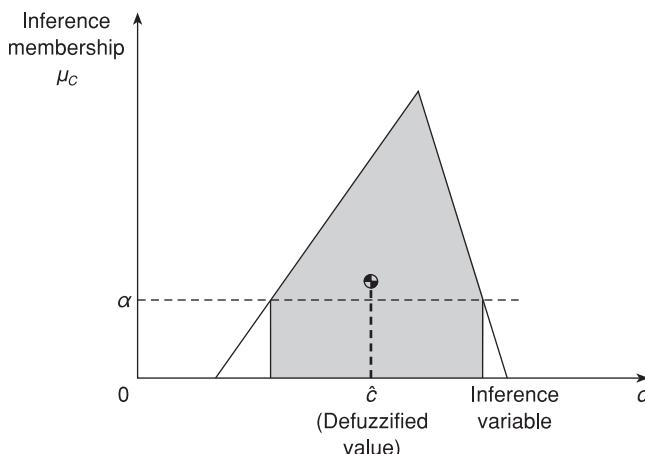


Figure 3.5: Thresholded defuzzification

membership function. As a result it is a more complete representation (first moment) of the inference. However, due to inherent averaging over the support set, the control action is diluted and less sensitive to small variations. Accordingly, the centroid method represents a very robust procedure, which generates less oscillatory (less chatter) process responses.

The *mean of maxima method* depends only on the peak values of the control inference membership function. It is therefore independent of the membership function shape, and particularly its asymmetry. Consequently, it is a somewhat incomplete representation of the control inference. The resulting control actions tend to be more sensitive and sharper in this method as compared to those from the centroid method. This is so because a shift of the peak position generally changes the centroid position continuously or by small increments, whereas the peak point of a membership function can shift by a substantial amount at a given time as a result of a change in shape of the membership function. It follows that the mean of maxima approach is more sensitive to small variations and is less robust. Furthermore, this method is rather insensitive to the fuzziness of the membership function, because the peak value and not the shape is used in the defuzzification process.

Thresholding, using the  $\alpha$ -cut given by equation (3.16) for the region of integration, will make the centroid method more sensitive, less robust, and less dependent on the shape of the control membership function. In the limiting case of  $\alpha = 1$ , however, there is no difference between the centroid method and the mean of maxima method. Note that in the mean of maxima method, when the membership grades at the modal points of the control inference are large (close to 1.0), thresholding has no effect on the defuzzified value.

### 3.3.4 Fuzzification

Fuzzification refers to the representation of a crisp value by a membership function. As seen before, this is needed prior to applying the composition (CRI), when the data (measurements) are crisp values, as common in control applications. In the example of Figure 3.4, measured value of temperature is fuzzified simply by picking the grade value of the membership function of the fuzzy variable temperature ( $T$ ) corresponding to the particular measured value. This is the method of fuzzy singleton, and is perhaps the most common method of fuzzification.

It may be argued that the process of fuzzification amounts to giving up the accuracy of crisp data. This is not so in general. The reason is, a measured piece of data may not be known to be 100% accurate. If the measured variable has some inherent fuzziness (similar to inherent randomness), it is particularly appropriate to assign a grade of membership (similar to assigning a probability) based on some prior knowledge regarding the measured quantity. This prior knowledge is available in the form of a membership function in the case of a fuzzy variable (similar to a probability distribution function in the case of a random variable).

### 3.3.4.1 Singleton method

Consider a crisp measurement  $y_0$  of a fuzzy variable  $Y$ . If it is known that the measurement is perfectly accurate, it may be represented by a fuzzy quantity  $F$  with the singleton membership function

$$\mu_F(y) = \delta(y - y_0) \quad (3.17)$$

where the familiar delta function is given by

$$\begin{aligned} \delta(y - y_0) &= 1 \quad \text{when } y = y_0 \\ &= 0 \quad \text{elsewhere} \end{aligned}$$

as in equation (3.6).

Since the measured data are not perfectly accurate, a more appropriate method of using fuzzy singleton to fuzzify a crisp value is given now. As before, suppose that a crisp measurement  $y_0$  is made of a fuzzy variable  $Y$ . Let it be known that  $Y$  can assume  $n$  fuzzy states  $Y_1, Y_2, \dots, Y_n$ . Since  $Y = Y_1 \text{ OR } Y_2 \text{ OR } \dots \text{ OR } Y_n$ , the membership function of  $Y$  is given as the union of the membership functions of the individual fuzzy states:  $\mu_Y(y) = \max_{j=1}^n \mu_{Y_j}(y)$ .

Then, the membership function of the fuzzified quantity  $F$  is given according to the extended singleton method by a set of fuzzy singletons where, for state  $j$ , the fuzzified quantity is:

$$\mu_F(y) = \mu_{Y_j}(y) \delta(y - y_0) \quad (3.18)$$

### 3.3.4.2 Triangular function method

Instead of a singleton, an appropriately scaled (shaped) triangular membership function may be used to represent the fuzzified quantity for each fuzzy state, similar to what is given in equation (3.18). Specifically, note that a triangular membership function (continuous case) may be expressed as

$$\begin{aligned} \mu_A(y) &= 1 - \frac{|y - y_0|}{s} \quad \text{for } |y - y_0| \leq s \\ &= 0 \quad \text{otherwise.} \end{aligned} \quad (3.19)$$

Note that  $y_0$  is the peak point, where the membership grade is 1, and  $s$  is the base length (support set). For example, a sharper peak is realized by making  $s$  smaller.

Again, suppose that a crisp measurement  $y_0$  is made of a fuzzy variable  $Y$ , which can assume  $n$  fuzzy states  $Y_1, Y_2, \dots, Y_n$ . Then, the membership function of the fuzzified quantity  $F$  is such that, for state  $j$ , the membership function of the fuzzified quantity is given by:

$$\mu_F(y) = \mu_{Y_j}(y_0) \mu_A(y) \quad (3.20)$$

where

$$\begin{aligned}\mu_{A_j}(y) &= 1 - \frac{|y - y_0|}{s_j} \quad \text{for } |y - y_0| \leq s_j \\ &= 0 \quad \text{otherwise.}\end{aligned}$$

The parameters  $s_j$  may be chosen so as to give different levels of fuzziness to the fuzzified values for different states. Note that  $s_j = 0$  corresponds to the limiting case of fuzzy singleton.

### 3.3.4.3 Gaussian function method

In the previous approach of fuzzification, the *shaping function* of the fuzzified membership function need not be triangular and can be quite general. The triangular shape is quite common, however. Another appropriate shaping function is given by the Gaussian membership function:

$$\mu_A(y) = \exp\left[\frac{y - y_0}{s}\right]^2 \quad (3.21)$$

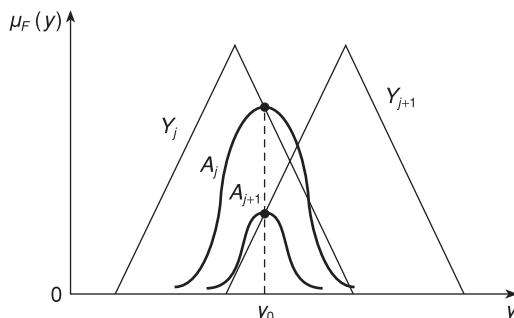
As before, the smaller the  $s$  the sharper (or less fuzzy) the membership function represented by equation (3.21). In the present method of fuzzification, the membership function of the fuzzified quantity  $F$  is such that, for state  $j$ , the membership function of the fuzzified quantity is given by:

$$\mu_F(y) = \mu_{Y_j}(y_0) \mu_{A_j}(y)$$

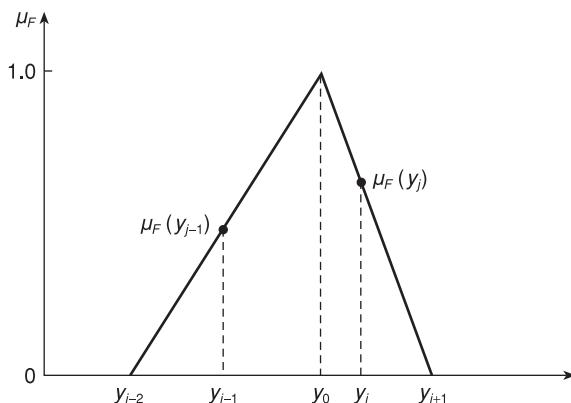
where

$$\mu_{A_j}(y) = \exp\left[\frac{y - y_0}{s_j}\right]^2$$

This approach is illustrated in Figure 3.6.



**Figure 3.6:** Fuzzification using a Gaussian shape function



**Figure 3.7:** Fuzzification using a discrete membership function

#### 3.3.4.4 Discrete case of fuzzification

The formulae of fuzzification given thus far are applicable to continuous membership functions. In the case of discrete membership functions, the crisp quantity  $y_0$  may not correspond to one of the discrete points of the membership function of the fuzzy variable Y (or fuzzy state  $Y_j$ ). In this case, the continuous method needs to be modified appropriately. One method, which may be further generalized, is given below.

Suppose that the crisp data value  $y_0$  falls between the discrete values  $y_{j-1}$  and  $y_j$  of the membership function, as shown in Figure 3.7. Assigning a membership grade of 1 for  $y_0$ , the membership grades of the fuzzified quantity F at  $y_{j-1}$  and  $y_j$  are determined through linear interpolation as:  $\frac{y_{j-1} - y_{j-2}}{y_0 - y_{j-2}}$  and  $\frac{y_{j+1} - y_j}{y_{j+1} - y_0}$ , respectively. Accordingly, in the notation introduced in Chapter 2, the discrete membership function of the fuzzified quantity F is given by:

$$F = \frac{y_{j-1} - y_{j-2}}{y_0 - y_{j-2}} / y_{j-1}, \quad \frac{y_{j+1} - y_j}{y_{j+1} - y_0} / y_j \quad (3.22)$$

The approach presented here can be extended in a straightforward manner to include more than two discrete points, thereby providing a wider membership function (greater fuzziness) simply by including additional points in the interpolation. Furthermore, nonlinear interpolation may be used as well, which corresponds to using a different shape function (e.g., Gaussian in place of triangular) in the case of fuzzification with continuous membership functions.

#### 3.3.5 Fuzzy control surface

A fuzzy controller is a nonlinear controller. A well-defined problem of fuzzy control, with analytical membership functions and fuzzification and

defuzzification methods, and well-defined fuzzy logic operators, may be expressed as a nonlinear control surface through the application of the compositional rule of inference. The advantage then is that the generation of the control action becomes a simple and very fast step of reading the surface value (control action) for given values of crisp measurement (process variables). The main disadvantage is the controller is fixed and cannot accommodate possible improvements to control rules and membership functions through successive learning and experience. Nevertheless, this approach to fuzzy control is quite popular.

### Example 3.2

A schematic diagram of a simplified system for controlling the liquid level in a tank is shown in Figure 3.8(a). In the control system, the error (actually, correction) is given by

$$e = \text{Desired level} - \text{Actual level}$$

The change in error is denoted by  $\Delta e$ . The control action is denoted by  $u$ , where  $u > 0$  corresponds to opening the inflow valve and  $u < 0$  corresponds to opening the outflow valve. A low-level direct fuzzy controller is used in this control system, with the control rule base as given in Figure 3.8(b).

The membership functions for  $E$ ,  $\Delta E$ , and  $U$  are given in Figure 3.8(c). Note that the error measurements are limited to the interval  $[-3a, 3a]$  and the  $\Delta$ error measurements to  $[-3b, 3b]$ . The control actions are in the range  $[-4c, 4c]$ .

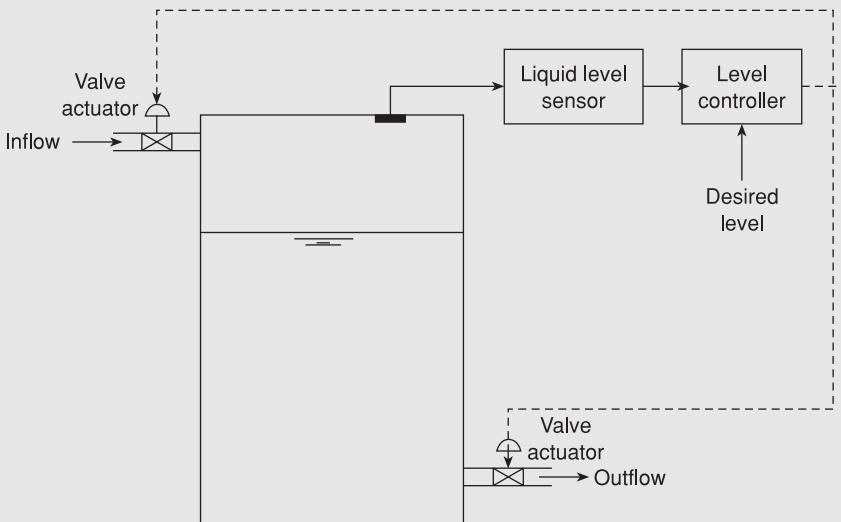


Figure 3.8 (a): Liquid level control system

$\Delta E$	NL	NS	ZO	PS	PL	
E	NL	NL	NM	NS	ZO	
	NS	NL	NM	NS	ZO	PS
	ZO	NM	NS	ZO	PS	PM
	PS	NS	ZO	PS	PM	PL
	PL	ZO	PS	PM	PL	PL

Figure 3.8 (b): The control rule base

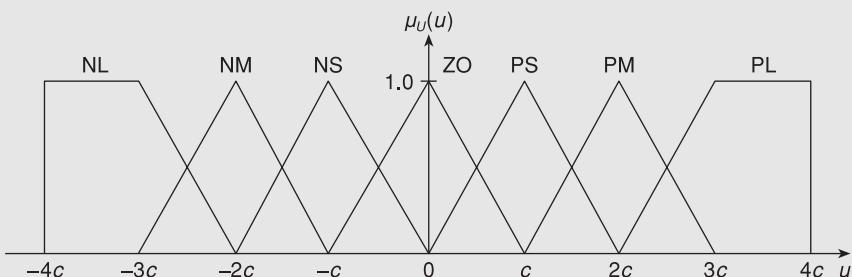
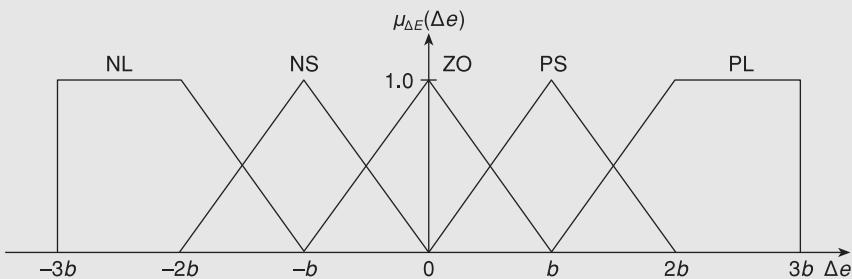
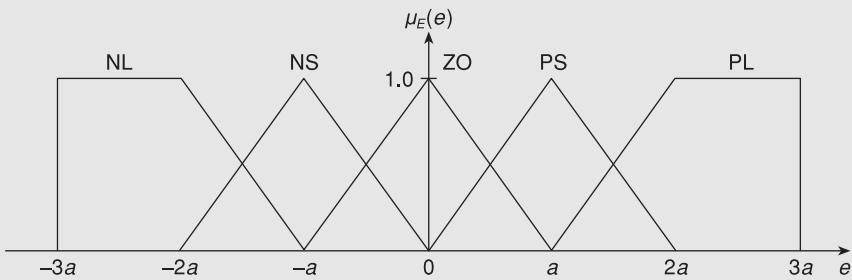


Figure 3.8 (c): The membership functions of error, change in error, and control action

Following the usual steps of applying the compositional rule of inference for this fuzzy logic controller, develop a crisp *control surface*  $u(e, \Delta e)$  for the system, expressed in the three-dimensional coordinate system  $(e, \Delta e, u)$ , which then can be used as a simple and fast controller.

### **Solution**

The crisp control surface is developed by carrying out the rule-based inference for each point:  $(e, \Delta e)$  in the measurement space  $E \times \Delta E$ , using individual rule-based inference. Specifically

$$\mu_{U'}(u) = \max_{i,j} \min[\mu_{E_i}(e_0), \mu_{\Delta E_j}(\Delta e_0), \mu_{U_k}(u)] \quad (3.23)$$

where

- $\mu_{U'}$  control inference membership function
- $e_0$  crisp context variable “error” defined in  $[-3a, 3a]$
- $\Delta e_0$  crisp context variable “change in error” defined in  $[-3b, -3b]$
- $E_i$  fuzzy states of “error”
- $\Delta E_j$  fuzzy states of “change in error”
- $U_k$  fuzzy states of “control action”
- $(i, j, k)$  possible combinations of fuzzy states of error, change in error, and control action, within the rule base.

To find the crisp control inference ( $u'$ ) for a set of crisp context data  $(e, \Delta e)$ , the fuzzy inference  $\mu_{U'}(u)$  is defuzzified using the center of gravity (centroid) method, which for the continuous case is:

$$u' = \frac{\int_{u \in U} u \mu_{U'}(u) du}{\int_{u \in U} \mu_{U'}(u) du} \quad (3.24a)$$

or for the discrete case, it is:

$$u' = \frac{\sum_{u_i \in U} u_i \mu_{U'}(u_i) du}{\sum_{u_i \in U} \mu_{U'}(u_i) du} \quad (3.24b)$$

where  $U = [-4c, 4c]$ . Also, if the geometric shape of the inference is simple (e.g., piecewise linear), the centroid can be computed by the moment method; thus

$$u' = \frac{\sum_{i=1}^n area_i m_i}{\sum_{i=1}^n area_i} \quad (3.24c)$$

where

- $\text{area}_i$  = area of the  $i$  th sub-region
- $m_i$  = distance of the centroid of the  $i$ -th sub-region, on the control axis.

To demonstrate this procedure, consider a set of context data  $(e_0, \Delta e_0)$ , where  $e_0$  is in  $[-3a, -2a]$  and  $\Delta e_0$  is in  $[-b/2, 0]$ . Then, from the membership functions and the rule base, it should be clear that only two rules are valid in this region, as given below:

- $R_1$ : if  $e$  is  $NL$  and  $\Delta e$  is  $NS$  then  $u$  is  $NL$
- $R_2$ : if  $e$  is  $NL$  and  $\Delta e$  is  $ZO$  then  $u$  is  $NM$

Since, in the range  $[-3a, -2a]$ , the membership grade of singleton fuzzification of  $e_0$  is always 1, the lower grade of the two context values is the one corresponding to the singleton fuzzification of  $\Delta e_0$  for both rules. Then, in applying the individual rule-based inference, the lower grade value of the two context variables is used to clip off the corresponding membership function of the control action variable  $U$  in each rule (this is a  $\min$  operation). The resulting membership functions of  $U$  for the two applicable rules are superimposed (this is a  $\max$  operation) to obtain the control inference  $U'$ , as shown in Figure 3.9.

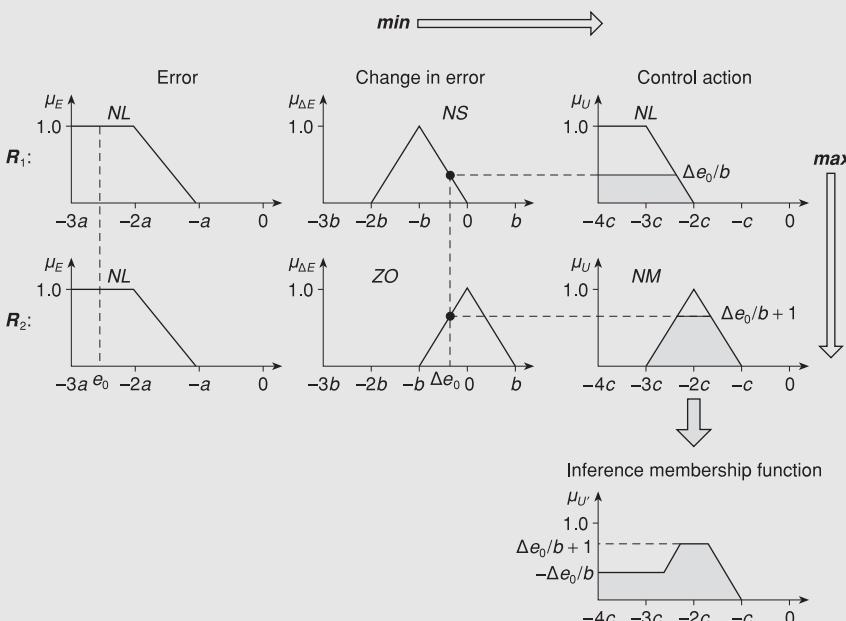


Figure 3.9: Individual rule-based inference for  $e_0[-3a, -2a]$  and  $\Delta e_0[-b/2, 0]$

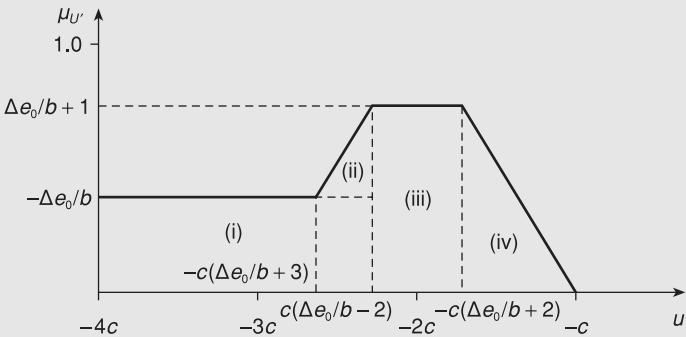


Figure 3.10: Sub-regions and critical points for calculation of the centroid

For defuzzification, we apply the moment method to find the centroid of the resulting membership function of control inference, as shown in Figure 3.10. Note that the critical points in Figure 3.9 and Figure 3.10 (e.g.,  $-\Delta e_0/b$ ,  $-c(\Delta e_0/b + 3)$ , etc.) are found from the corresponding membership functions.

From the moment method, we obtain the crisp control action as a function of  $e$  and  $\Delta e$ . The above procedure is repeatedly applied to all possible ranges of  $e[-3a, 3a]$  and  $\Delta e[-3b, 3b]$ , to obtain the complete control surface. Also, the procedure can be implemented in a computer program to generate a control surface. A control surface with  $a = 1$ ,  $b = 2$ , and  $c = 0.5$  is shown in Figure 3.11.

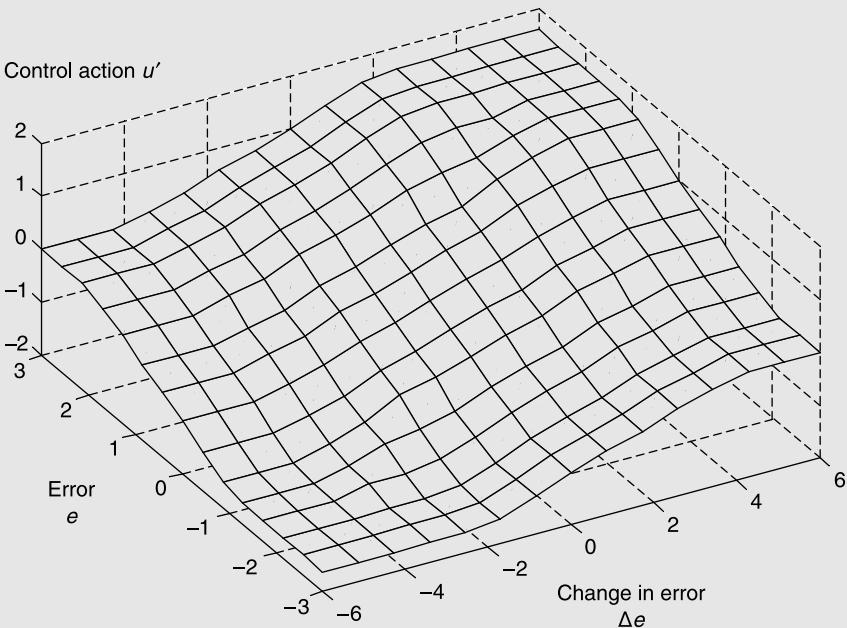


Figure 3.11: Control surface with  $a = 1$ ,  $b = 2$ , and  $c = 0.5$

### 3.3.6 Extensions of Mamdani fuzzy control

In the present chapter what we have presented as direct fuzzy control is in fact the Mamdani approach (Mamdani system or Mamdani model) named after the person who pioneered the application of this approach. Here, the knowledge base is represented as fuzzy protocols of the form expressed by equation (3.1) and represented by a membership function of the form given by (3.3), and the control inference is obtained by applying the compositional rule of inference according to equation (3.4). Several variations to this conventional fuzzy logic control (FLC) are available. One such version is the Sugeno model (or Takagi–Sugeno–Kang model or TSK model). Here, the knowledge base has fuzzy rules with crisp functions as the inferences, of the form

$$\text{IF } x \text{ is } A_i \text{ AND IF } y \text{ is } B_i \text{ THEN } c_i = f_i(x, y) \quad (3.25)$$

for Rule  $i$ , where,  $f_i$  is a crisp function of the condition variables (process outputs)  $x$  and  $y$ . Note that the condition part of this rule is the same as for the Mamdani model (3.1), where  $A_i$  and  $B_i$  are fuzzy sets whose membership functions are functions of  $x$  and  $y$ , respectively. Unlike in the Mamdani approach, the action part is a crisp function of the condition variables in the TSK approach. The control inference  $\hat{c}(x, y)$  is obtained directly as a crisp function of the condition variables  $x$  and  $y$ , as follows:

For Rule  $i$ , a weighting parameter  $w_i(x, y)$  is obtained corresponding to the condition membership functions, as for the Mamdani approach, by using either the “min” operation or the “product” operation. For example, using the “min” operation we form

$$w_i(x, y) = \min[\mu_{A_i}(x), \mu_{B_i}(y)] \quad (3.26)$$

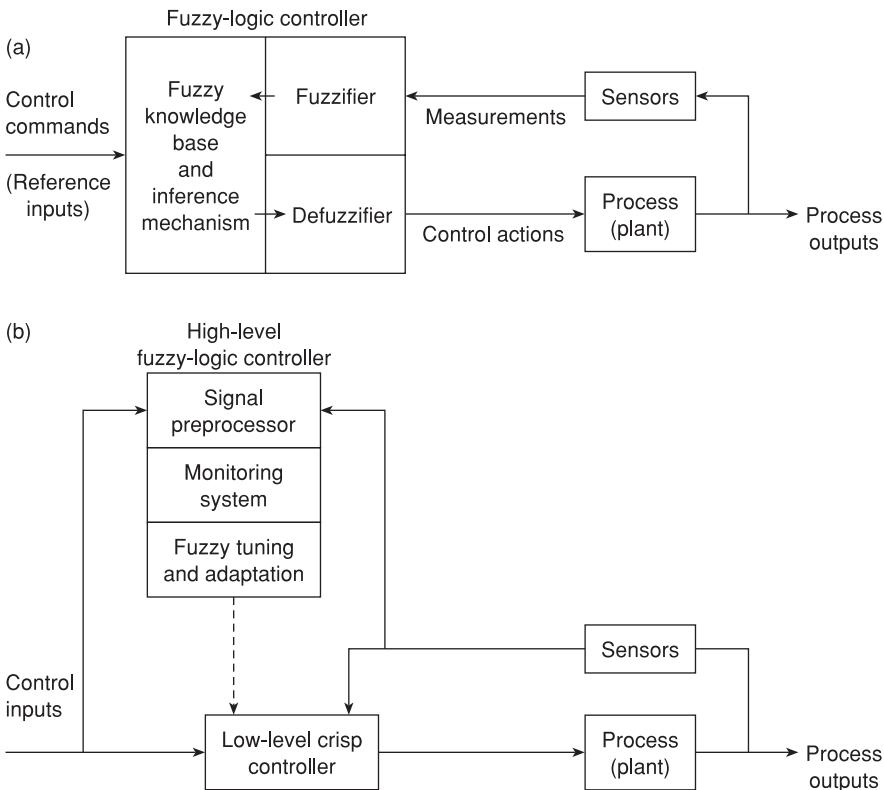
The crisp control inference  $\hat{c}(x, y)$  is determined as a weighted average of the individual rule inferences (crisp)  $c_i = f_i(x, y)$  according to

$$\hat{c}(x, y) = \frac{\sum_{i=1}^r w_i c_i}{\sum_{i=1}^r w_i} = \frac{\sum_{i=1}^r w_i(x, y) f_i(x, y)}{\sum_{i=1}^r w_i(x, y)} \quad (3.27)$$

where  $r$  is the total number of rules. For any output measurements  $x$  and  $y$ , the control action  $\hat{c}(x, y)$  can be computed from (3.27), without requiring any defuzzification. The Sugeno model is particularly useful when the control actions are described analytically through crisp functions, as in conventional crisp control, rather than linguistically.

## 3.4 Fuzzy control architectures

Two architectures of fuzzy logic control are common and are shown in Figure 3.12. The conventional architecture, as shown in Figure 3.12(a) and



**Figure 3.12:** Architectures of fuzzy control. (a) Low-level direct control. (b) High-level supervisory control

illustrated before, is by far the most common one. Here the fuzzy logic controller generates the low-level direct control signals. This approach has some drawbacks with respect to speed, accuracy, and sensitivity and may not be appropriate unless some of the following conditions are satisfied:

- (1) Process time constants are relatively high (e.g., 0.1 seconds or more).
- (2) The required control bandwidth is sufficiently low (e.g., 10.0 Hz or less).
- (3) The process is complex and ill-defined. Analytical modeling is difficult and experimental model identification is infeasible.
- (4) Sufficient past experience with low-level human-in-the-loop control is available and may be expressed as a set of linguistic protocols with fuzzy terms.

A more desirable architecture, when these constraints are violated, is the hierarchical structure as shown in Figure 3.12(b). This is a supervisor control architecture where low-level direct control is done using conventional crisp techniques, while supervisory tasks such as process monitoring, performance assessment, tuning, adaptation, and restructuring are done by upper levels. In

particular, fuzzy decision-making is appropriate in performance assessment and controller tuning.

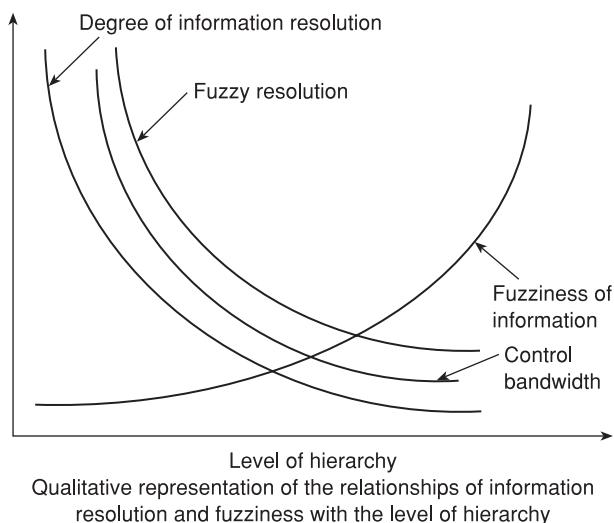
The control system of a complex plant can benefit enormously by using a suitable, hierarchical architecture. Consider a process plant consisting of one or more workcells. Each workcell may consist of components such as robots, material transfer modules, processing devices, fixturing, and monitoring systems, with associated controllers, sensors, and actuators. The components have to be properly interfaced or linked for automated and autonomous operation of the system. The performance of the system will strongly depend on the control and communication architecture that is employed. In particular, a control system consisting of a hierarchical architecture would be appropriate. Specifically, the system may be arranged into a functional hierarchy, with modules (both hardware and software) of similar functionality occupying a single layer. Each module will possess a fixed input–output structure, and will represent an object in an object-oriented paradigm. Various tasks that are performed by the plant have to be identified and planned. The tasks will define the necessary hardware and software components (e.g., material transfer units, sensors, actuators, signal conditioners, and controllers) and can be represented in a modular manner, as a set of *objects*, each object having a fixed input–output structure. Since the tasks have to be performed according to some hierarchy, the objects can be linked according to a hierarchical architecture. Each layer may be considered as a knowledge-based system. Typically the complexity of decision-making will increase with the hierarchical level, and the level of intelligence that is needed to make the decisions will increase as well. At least some of the upper layers will contain knowledge systems for carrying out high-level control tasks. In particular, a fuzzy logic controller may operate at a higher level, to perform tuning and adaptive control functions rather than in-loop direct control. Since high-level decision-making requires high-level information, an important component in the control system is a signal preprocessor. In particular, high-resolution information from sensors in the lower levels has to be properly “filtered” so as to provide compatible context for decision-making through the upper-level knowledge systems. The resulting (preprocessed) information is generally fuzzier and has a lower resolution. In this architecture, a monitoring system uses high-level process information to evaluate the performance of the plant. On this basis, a fuzzy decision-making unit is employed to make adjustments to the control system. These adjustments may include on line and off line tuning of the controller parameters, on line adaptation of the low-level controller, and self-organization and dynamic restructuring of the control system.

### 3.4.1 Hierarchical fuzzy systems

Since the upper layers of a hierarchical architecture generally deal with low-resolution, imprecise, and incomplete information, more intelligence (or knowledge-based action) would be needed in the associated decision-making process. Yet the performance of the overall system may be acceptable. This is a characteristic of an intelligent system. Task description, knowledge

representation, and decisions at high levels of a hierarchy may be qualitative and linguistic and hence fuzzy, and consequently some intelligence will be needed for interpretation and processing of this information in order to make inferences (and control actions). Information that is qualitative, fuzzy, vague, ambiguous, general, imprecise, and uncertain may be present at higher levels of control, and is also amenable to intelligent controllers. On the other hand, at low levels, a controller will typically need crisp, precise and non-fuzzy information (reference inputs, feedback signals, parameter values, etc.); otherwise the system performance can deteriorate considerably.

Another distinction between various levels of a hierarchical system can be made in terms of the cycle time of the associated events and their control bandwidth. Due to the nature, amount, and complexity of the information that is needed at higher levels, the computational overheads will be higher and the speed of decision-making will be slower as well. This is not a serious drawback because the cycle times of the events at higher levels of a hierarchical control system are generally long. In contrast, at low levels of a hierarchy, faster action with short-term information will be needed as dictated by the process speed requirements and as a result, the control bandwidth will be higher. Information of high resolution can be intractable at high levels of a hierarchy due to the fact that for a specified amount of information that is required at a particular level, the amount of data needed will directly (and in many cases, exponentially) increase with the degree of resolution. Indeed, high levels of resolution may not be needed at upper hierarchical levels. Quite the opposite is generally true at low levels. Figure 3.13 summarizes the qualitative relationships of information resolution, control bandwidth, and fuzziness with the level of control hierarchy. The present section will formalize several of these considerations.



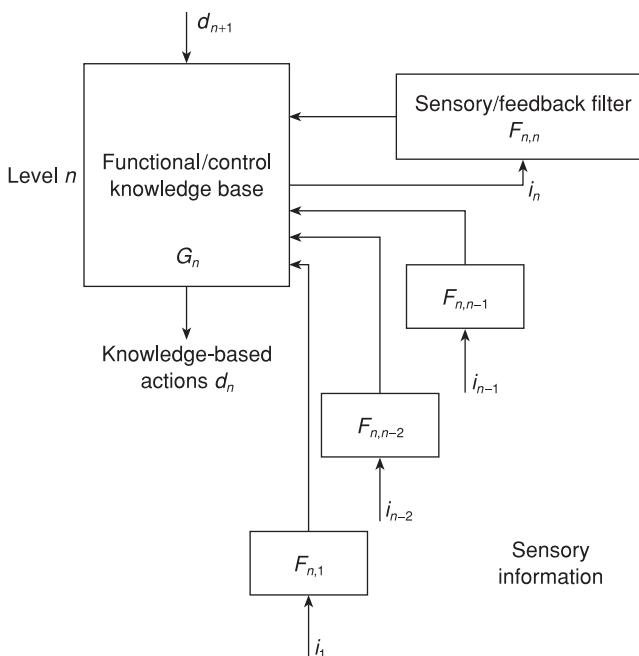
**Figure 3.13:** Relationships of information resolution and fuzziness with respect to hierarchical level

### 3.4.2 Hierarchical model

Procedural descriptions of tasks and subtasks at the top level of a hierarchy may be done in a linguistic form. For example, statements such as “High-grade filleting for export” may contain fuzzy, qualitative, vague, general, uncertain and imprecise terms and interpretations. In particular, a high level of fuzziness may be tolerated at high levels of the control hierarchy, while even a slight fuzziness at the lowest level can lead to inaccurate control. In this context the meanings of *fuzzy resolution* and *degree of fuzziness* are important, which are discussed in the chapter on fuzzy logic. As the fuzzy resolution increases, the information resolution also increases, and the corresponding knowledge base becomes more appropriate for a lower level of the control hierarchy.

Consider a hierarchical control structure. Our emphasis here is on the knowledge-based control aspects of the system, including controller tuning and decision-making at higher levels. First it is assumed that the overall, hierarchical system has been designed first using conventional techniques. This includes task allocation, resource allocation, system optimization, and low-level control. Our objective then is to develop a supplemental hierarchical system that uses additional knowledge and experience to further improve the performance of the system. In particular, knowledge that may be expressed in a linguistic and qualitative form and which is amenable to fuzzy logic processing is important here.

An analytical model for a fuzzy-knowledge based, hierarchical control system is shown in Figure 3.14. What is shown is a general level (level  $n$ ) of



**Figure 3.14:** Schematic representation of a hierarchical model

the knowledge-based structure. This particular structure of information transfer and control can have a variety of applications. Functional modules are shown in the figure as blocks, and the signal and communication paths are indicated as directed lines. In particular, we observe that there are “action lines” corresponding to inferences, commands and dynamic reference levels, which go from an upper level to a lower level, and “sensing lines” which are lateral towards the sensory/feedback blocks which produce feedback signals.

The “feedback lines” can occur both at the same level and from a lower level to an upper level. In the latter case, a filter/interpreter is used to convert the information into a form that is consistent with the data acquisition capabilities of the upper level. This may involve a process of information abstraction. The notation  $G$  is used to denote the functional and control knowledge base at a particular level of the hierarchy, and it contains structured information in the form of linguistic fuzzy rules. The knowledge base receives instructions and inferences  $c$  from the upper level and also information and specialized knowledge  $s$  from sensory feedback and knowledge sources at the present level and the lower level. When sensory information is received from the lower level, an information filter, preprocessor, or an interpreter (denoted by  $F$ ) will be necessary in order to generate the corresponding feedback commands. This hierarchical structure is operationally expressed as

$$c_n = G_n \otimes [(F_{n,n} \otimes s_n) \oplus (F_{n,n-1} \otimes s_{n-1}) \oplus \dots \oplus (F_{n,1} \otimes s_1) \oplus c_{n+1}] \quad (3.28)$$

for level  $n$

in which

$F_{i,j}$  = information prefilter from Level  $j$  to Level  $i$

$s_j$  = information vector from Level  $j$

$G_i$  = knowledge base at Level  $i$

$c_i$  = decision vector at Level  $i$

$\oplus$  = a combinational operator

$\otimes$  = a transitional operator

The knowledge bases  $G_i$  and  $F_i$  may take linguistic, fuzzy logic representations. Also, the operations  $\oplus$  and  $\otimes$  which are subjective and have interpretations depending on the specific application may be based on various fuzzy logic operations and the compositional rule of inference. In particular these operations can have a direct influence on the fuzziness of various information components that are being preprocessed and combined.

Since intelligence may be considered as the capacity to acquire and meaningfully apply knowledge within the context of perception, reasoning, learning and inference from incomplete knowledge, it is directly related to the interpretation of the transitional operator  $\otimes$  and the combinational operator  $\oplus$ . In the context of fuzzy inference, then, these operators can be interpreted within the framework of the compositional rule of inference, for example using the *sup-min composition*.

### 3.4.2.1 Feedback/filter modules

In Figure 3.14, the feedback filter (preprocessor) modules are denoted by  $F$ . The nature of information processing that is required in each such module will depend on several factors such as the level and the level differential, which the  $F$  module serves. Generally, when the level is high and the level differential is high, high-level information processing will be involved. The cycle times of processing will be longer and, furthermore, information interpretation will become more prominent than direct processing. Formally, the transitional operation ( $\otimes$ ) of the feedback filter, which is given by:

$$\bar{s} = F \otimes s \quad (3.29)$$

may mean

$$\bar{s}(\bar{t}) = \bar{s}(s[\bar{t}, \Delta T]) \quad (3.30)$$

with

$$s = [s_1, s_2, \dots, s_n]^T \quad (3.31)$$

$$\bar{s} = [\bar{s}_1, \bar{s}_2, \dots, \bar{s}_m]^T \quad (3.32)$$

and  $m \leq n$ . Here  $\Delta T$  denotes the cycle time of the operation, which represents the duration of the input information  $s$  that is needed to generate one instant of inferences (outputs)  $\bar{s}$  within the  $F$  module. Note that  $\bar{s}$  represents part of the context that is needed by the functional control module  $G$  at the particular level. The number of inference channels  $m$  of  $F$  is generally less than the number of input information channels  $n$  in high-level processing.

#### Example 3.3

Inputs to a feedback/filter module for Level 2 of a fish processing workcell may include cutter blade speed, conveyor speed and images of processed fish over a duration of 60 s. The output inference made by  $F$  might be a single “quality index” of processing for over 100 fish that are processed during this period. Peak variations, average values and shape factors may be involved in establishing the quality index “value.” The value itself may be binary, of the type “acceptable/unacceptable.”

Since the membership grades of the input information to  $F$  are processed through “*min*” type norms corresponding to “AND” or cross-product operators, it should be clear that the inferences of  $F$  will generally be fuzzier than the input information. To further illustrate this, consider a “data averaging” function. Then, assuming that the average “mapping” itself is crisp and the data set alone is fuzzy, we may represent the transitional operator  $\otimes$  by

$$\bar{s}_j(\bar{t}) = \frac{1}{T} \int_{\tau}^{\bar{t}+T} s_j(\tau) d\tau \quad (3.33)$$

Then, the membership grade of the filter output is given by

$$\mu_{\bar{s}_j(\tau)} \min_{[\bar{t}, \bar{t}+T]} [\mu_{s_j(\tau)}] \quad (3.34)$$

This aspect of information preprocessing will be further discussed later.

### 3.4.2.2 Functional/control modules

In Figure 3.14, the functional and control modules are denoted by  $G$ . In the present model, each module  $G$  contains a linguistic knowledge base, and makes decisions (actions)  $c$  using the context as determined by the decisions of the upper level as well as inferences made by the related feedback/filter modules. Then, equation (3.28) may be rewritten as

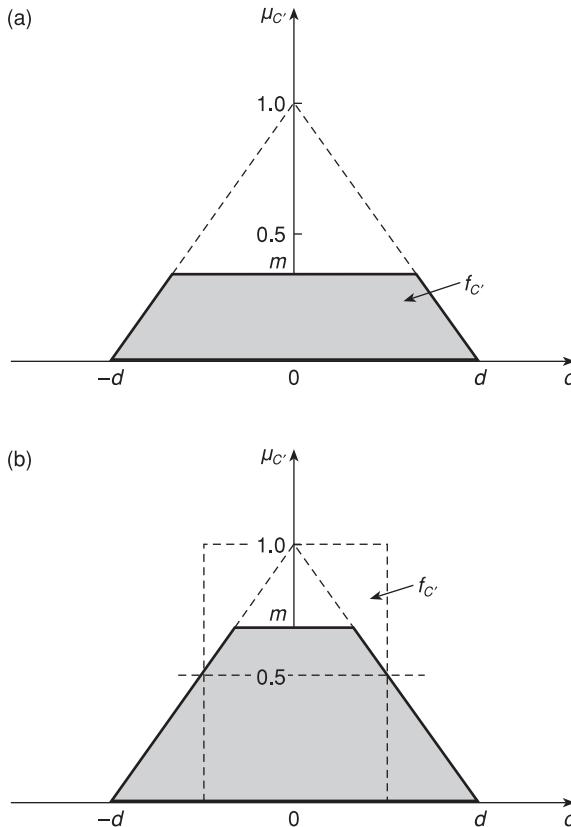
$$c_n = G_n \otimes [\bar{s}_n \oplus \bar{s}_{n-1} \oplus \dots \oplus \bar{s}_1 \oplus c_{n+1}] \quad (3.35)$$

Here the combinational operator will take appropriate interpretations depending on the context information that is needed by the functional control module and the inferences that are provided by the feedback/filter modules. For example, it may mean a “*sup*” operation for membership functions of like elements in various vectors on the right-hand side of equation (3.35). Generally then, the control decisions and actions  $c$  are established by applying the compositional rule of inference. The combinational operation is further discussed below, in the context of information processing.

The upper-level functional/control modules  $G$  will have learning and self-organization capabilities as well. For example, those membership functions corresponding to decisions/actions that apparently produce a worsening performance can be tagged. Then the tagging statistics may be used to modify these membership functions and even to delete the corresponding linguistic rules altogether from the knowledge base. Similarly, the inferences that tend to improve the system performance may be tagged as well and subsequently used to enhance the corresponding rules and membership functions. Furthermore, new rules may be added to the knowledge base on the basis of experience and new knowledge. The functional/control module  $G$  need not be complete, for the purpose of the supplemental knowledge system. At higher levels, it should be able to operate at a high level of intelligence with fuzzier, incomplete, and approximate information of relatively low resolution.

### 3.4.3 Effect of information processing

In this section, we explore the effect of preprocessing (transitional operation  $\otimes$ ) on the fuzziness of the result. This has direct implications in hierarchical control systems, as low-level information is preprocessed for use in high-level decision-making, as discussed in connection with the model given in Figure 3.14.



**Figure 3.15:** Control inference and its  $1/2$ -cut when (a) the context membership is  $< 0.5$ , (b) the context membership is  $\geq 0.5$

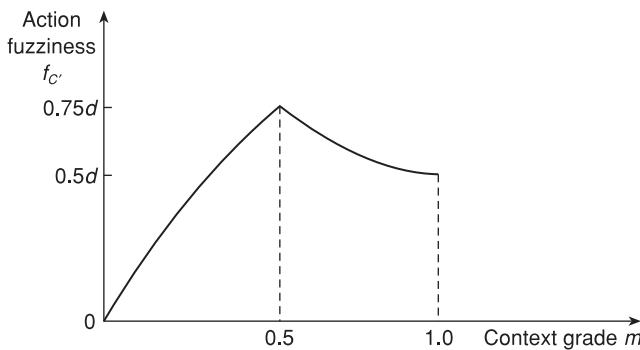
Suppose that the membership function  $\mu_c$  of the control action  $C$  is triangular and symmetric, with a unity modal value and a support set of  $[-d, d]$ , as shown in Figure 3.15. Also, suppose that a set of low-level signals is measured and fuzzified to give membership grades  $m_i$  and that

$$m = \min_i(m_i) \quad (3.36)$$

If the knowledge base uses the fuzzy quantities corresponding to  $m_i$  as the context, then the control inference for these measurements is obtained by applying the compositional rule. Clearly, the shaded area in Figure 3.15 corresponds to the control inference  $C'$ .

To obtain the fuzziness of  $C'$ , we need to determine the  $1/2$ -cut of  $C'$ . Two cases exist:

- (a)  $m < 0.5$ , as shown in Figure 3.15(a)
- (b)  $m \geq 0.5$ , as shown in Figure 3.15(b).



**Figure 3.16:** Variation of the fuzziness of a control inference with the membership grade of the rule base context

It should be clear in case (b) that  $\mu_{C'_{1/2}}$  is the rectangular pulse shown in Figure 3.15(b). Then using the measure of fuzziness  $f_{C'} = \int_{-d}^d |\mu_C(x) - \mu_{C'_{1/2}}(c)| dc$ ,

the following result is obtained:

$$\begin{aligned} f_{C'} &= dm(2-m) && \text{for } m < 0.5 \\ &= \frac{d}{2}(3-4m+2m^2) && \text{for } m \geq 0.5 \end{aligned} \quad (3.37)$$

The relationship given by equation (3.37) is plotted in Figure 3.16. It is seen from this figure that the fuzziness of the control inference peaks at  $m = 0.5$ , which is the case when the minimum membership grade of the context variables is 0.5. In other words, the control inference is most fuzzy when the fuzzy state of the dominant sensor reading is also most fuzzy ( $\mu = 0.5$ ).

### 3.4.4 Effect of signal combination on fuzziness

Now consider decision-making with multiple rules and multiple contexts where the combinational operation  $\oplus$  would be applicable. The fuzziness of the control inference is given by

$$f_{C'} = \int_{c \in Z} |\mu_C(c) - \mu_{C'_{1/2}}(c)| dc \quad (3.38)$$

This can be written as

$$f_{C'} = \int_{c \in Z} |\max_i [\mu_{C'_i}(c)] - \max_j [\mu_{C'_{j/2}}(c)]| dc \quad (3.39)$$

Note that  $Z$  is the universe of discourse (or at least the support set) of  $C$ . Equation (3.39) follows from the fact that

$$\alpha\text{-cut}(A \cup B) = (\alpha\text{-cut } A) \cup (\alpha\text{-cut } B) \quad (3.40)$$

which may be easily verified. Next, in view of

$$|\max_i x_i - \max_j y_j| \leq \max_i |x_i - y_j| \quad (3.41)$$

we have

$$f_{C'} \leq \int_{\alpha \in Z} \max_i \mu_{C'_i}^f dc \quad (3.42)$$

Here  $\mu_{C'_i}^f$  is the local fuzziness at value  $c$  of the control inference  $C'_i$ , as given by

$$\mu_{C'_i}^f(c) = |\mu_{C'_i}(c) - \mu_{C'_{i/2}}(c)| \quad (3.43)$$

Hence equation (3.42) states that under the combinational operation, the fuzziness of the result is bounded by the envelope of the local fuzziness curves of the individual components of information.

### 3.4.5 Decision table approach for a fuzzy tuner

In this section we shall outline a fuzzy tuner for a proportional-integral-derivative (PID) servo controller, which takes the hierarchical architecture shown in Figure 3.12(b). First, expert knowledge on tuning a PID servo is expressed as a set of linguistic statements. Such statements (rules) will necessarily contain fuzzy quantities. Next, membership functions are established for the fuzzy quantities that are present in the fuzzy rules. Each fuzzy rule is then expressed as a discrete fuzzy relation, in tabular form, by incorporating the membership functions into it through fuzzy logic connectives (e.g., IF-THEN, OR). The resulting set of tables forms the rule base of the fuzzy tuner. Rule matching can be accomplished by on line application of the compositional rule of inference to the fuzzy rule base, if so desired. This on line application of the compositional rule is generally time consuming, and hence it may considerably reduce the speed of servo tuning. Since the universe of discourse of the condition set is discrete and finite in the present application, and since the universe of the tuning action is also discrete and finite, both classes of universes having low cardinality, computational efficiency can be significantly improved by applying the compositional rule of inference off line. This preprocessing procedure will generate a *decision table* for fuzzy tuning of a servo controller. The steps of developing a fuzzy decision table for servo tuning are outlined below.

Consider a set of linguistic statements given below. These linguistic rules reflect the actions of a human expert in tuning a PID servo, by observing the error response of the servo. Of course, more rules can be added, and the resolution (the number of fuzzy states) can be increased to improve the tuning accuracy. For the purpose of the present demonstration, however, this rule

	if	OSC = LOW	then	DP = NL
or	if	OSC = MOD	and then	DD = PL DP = NM
or	if	OSC = HIG	and then	DD = PM DP = NH
or	if	OSC = VHI	and then	DP = PH DI = NL
	if	RSP = LOW	and then	DD = PH DP = PM
or	if	RSP = MOD	and then	DD = PL DP = PL
or	if	RSP = HIG	and then	DP = NL DD = PL
or	if	RSP = VHI	and then	DP = NM DP = NL
	if	DIV = LOW	and then	DD = PL DD = NL
or	if	DIV = MOD	and then	DP = NL DD = PM
or	if	DIV = HIG	and then	DP = NL DI = NL
or	if	DIV = VHI	and then	DD = PH DP = NM
	if	OFS = LOW	and then	DI = NM DP = PL
or	if	OFS = MOD	and then	DD = PH DI = PL
or	if	OFS = HIG	and then	DI = PM DP = PL
or	if	OFS = VHI	and then	DI = PH DP = PM
			and	DI = PH

Figure 3.17: Condensed form of tuning protocols

set is adequate. We can express the fuzzy tuning rules of these linguistic protocols in the condensed form shown in Figure 3.17.

*If the response oscillations are low then slightly decrease the proportional gain and slightly increase the derivative time constant; or if the response is moderately oscillatory, then moderately decrease the proportional gain and moderately increase the derivative time constant; or if the response is highly oscillatory, then make a large decrease in the proportional gain and a large increase in the derivative time constant; or if the response is extremely oscillatory, then make a large decrease in the proportional gain*

*and a large increase in the derivative time constant and a slight decrease in the integral rate.*

*If the response is slow then increase the derivative time constant slightly and increase the proportional gain moderately; or if the speed of response is moderate then increase the derivative time constant slightly and increase the proportional gain slightly; or if the speed of response is high increase the derivative time constant slightly and decrease the proportional gain slightly; or if the speed of response is very high decrease the proportional gain moderately.*

*If the response diverges slowly then slightly decrease the proportional gain and slightly increase the derivative time constant; or if the response diverges moderately then slightly decrease the proportional gain and moderately increase the derivative time constant; or if the response diverges rapidly then slightly decrease the proportional gain and increase the derivative time constant by a large amount and slightly decrease the integral rate; or if the response diverges very rapidly then moderately decrease the proportional gain and increase the derivative time constant by a large amount and moderately decrease the integral rate.*

*If the offset is low then slightly increase the proportional gain, and slightly increase the integral rate; or if the offset is moderate then moderately increase the integral rate; or if the offset is high then increase the proportional gain slightly and increase the integral rate by a large value; or if the offset is very high then moderately increase the proportional gain and increase the integral rate by a large value.*

We need membership functions for the fuzzy conditions OSC, RSP, DIV and OFS and for the fuzzy actions DP, DI and DD. Several methods for estimating membership functions are available. For our purposes, an approximate set of membership functions would suffice, based on intuition. Generally there is freedom to choose the resolution of each fuzzy quantity. In the present example, however, since the condition variables are constrained to a discrete and finite set, the resolution of the condition variables would be fixed. For action variables, however, this restriction of resolution is not necessary. For example, even though the action variable DP can assume one of seven fuzzy values, the universe of say, NH, may contain more than seven elements. However, in the present demonstration, in the interest of consistency and simplicity, the cardinality of the universe of discourse of a fuzzy quantity is taken to be equal to the number of fuzzy values that can be assumed by the particular attribute.

In an application of the present type, actual numerical values given to the elements in a universe of a fuzzy quantity are of significance only in a relative sense. An appropriate physical meaning should be attached to each value. For instance, in defining DP, the numerical value  $-3$  is chosen to represent a negative high change in the proportional gain. This choice is compatible with the choice of  $-1$  to represent a negative low change. Also, such numerical values may be scaled and converted as necessary, to achieve physical and dimensional compatibility. The chosen membership functions are given in

**Table 3.1:** Membership functions of the condition variables**1. Oscillation**

OSC	0	1	2	3	4
NON	1.0	0.8	0.6	0.4	0.2
LOW	0.8	1.0	0.8	0.6	0.4
MOD	0.6	0.8	1.0	0.8	0.6
HIG	0.4	0.6	0.8	1.0	0.8
VHI	0.2	0.4	0.6	0.8	1.0

**2. Speed of response**

RSP	0	1	2	3	4
NON	1.0	0.8	0.6	0.4	0.2
LOW	0.8	1.0	0.8	0.6	0.4
MOD	0.6	0.8	1.0	0.8	0.6
HIG	0.4	0.6	0.8	1.0	0.8
VHI	0.2	0.4	0.6	0.8	1.0

**3. Divergence**

DIV	0	1	2	3	4
NON	1.0	0.8	0.6	0.4	0.2
LOW	0.8	1.0	0.8	0.6	0.4
MOD	0.6	0.8	1.0	0.8	0.6
HIG	0.4	0.6	0.8	1.0	0.8
VHI	0.2	0.4	0.6	0.8	1.0

**4. Offset**

OFS	0	1	2	3	4
NON	1.0	0.8	0.6	0.4	0.2
LOW	0.8	1.0	0.8	0.6	0.4
MOD	0.6	0.8	1.0	0.8	0.6
HIG	0.4	0.6	0.8	1.0	0.8
VHI	0.2	0.4	0.6	0.8	1.0

Tables 3.1 and 3.2. Note that, as desired, a membership grade of unity is assigned to the representative value of each fuzzy quantity. Fuzziness is introduced by assigning uniformly decreasing membership grades to the remaining element values.

Next we illustrate the development of a fuzzy relation table for a group of fuzzy rules. For example, consider the rule

$$\text{If OSC} = \text{LOW} \text{ then DP} = \text{NL} \quad (3.44)$$

To construct its fuzzy relation table, we take the membership function of LOW in OSC from Table 3.1 and the membership function of NL in DP from

**Table 3.2:** Membership functions of the action variables**1. Proportional gain**

DP	-3	-2	-1	0	1	2	3
NH	1.0	0.6	0.2	0	0	0	0
NM	0.6	1.0	0.6	0.2	0	0	0
NL	0.2	0.6	1.0	0.6	0.2	0	0
NC	0	0.2	0.6	1.0	0.6	0.2	0
PL	0	0	0.2	0.6	1.0	0.6	0.2
PM	0	0	0	0.2	0.6	1.0	0.6
PH	0	0	0	0	0.2	0.6	1.0

**2. Integral rate**

DI	-3	-2	-1	0	1	2	3
NH	1.0	0.6	0.2	0	0	0	0
NM	0.6	1.0	0.6	0.2	0	0	0
NL	0.2	0.6	1.0	0.6	0.2	0	0
NC	0	0.2	0.6	1.0	0.6	0.2	0
PL	0	0	0.2	0.6	1.0	0.6	0.2
PM	0	0	0	0.2	0.6	1.0	0.6
PH	0	0	0	0	0.2	0.6	1.0

**3. Derivative time constant**

DD	-3	-2	-1	0	1	2	3
NH	1.0	0.6	0.2	0	0	0	0
NM	0.6	1.0	0.6	0.2	0	0	0
NL	0.2	0.6	1.0	0.6	0.2	0	0
NC	0	0.2	0.6	1.0	0.6	0.2	0
PL	0	0	0.2	0.6	1.0	0.6	0.2
PM	0	0	0	0.2	0.6	1.0	0.6
PH	0	0	0	0	0.2	0.6	1.0

Table 3.2. Next, in view of the fact that fuzzy implication is a “min” operation on membership grades, we form the Cartesian product space of these two membership function vectors and assign the lower value of each pair of membership grades to the corresponding location in the Cartesian space. This corresponds to the formation of the discrete membership function of the particular rule (see equation (3.2)). The other three relation tables representing the remaining three rules (see Figure 3.17) that connect OSC to DP are obtained in a similar fashion. The corresponding four rules are connected by fuzzy OR connectives. Hence the composite (aggregate) relation table is obtained by combining the individual tables through a “max” operation (see equation (3.3)). The composite relation tables obtained in this manner are given in Table 3.3.

The final step in the development of the fuzzy tuner is the establishment of the decision table. Each inference of servo response (condition) has to be

**Table 3.3:** Composite fuzzy-relation tables

(a) OSC to DP

		DP						
		-3	-2	-1	0	1	2	3
OSC	0	0.6	0.6	0.8	0.6	0.2	0	0
	1	0.6	0.8	1.0	0.6	0.2	0	0
	2	0.8	1.0	0.8	0.6	0.2	0	0
	3	1.0	0.8	0.6	0.6	0.2	0	0
	4	1.0	0.6	0.6	0.4	0.2	0	0

(b) OSC to DI

		DI						
		-3	-2	-1	0	1	2	3
OSC	0	0.2	0.2	0.2	0.2	0.2	0	0
	1	0.2	0.4	0.4	0.4	0.2	0	0
	2	0.2	0.6	0.6	0.6	0.2	0	0
	3	0.2	0.6	0.8	0.6	0.2	0	0
	4	0.2	0.6	1.0	0.6	0.2	0	0

(c) OSC to DD

		DD						
		-3	-2	-1	0	1	2	3
OSC	0	0	0	0.2	0.6	0.8	0.6	0.6
	1	0	0	0.2	0.6	1.0	0.8	0.6
	2	0	0	0.2	0.6	0.8	1.0	0.8
	3	0	0	0.2	0.6	0.6	0.8	1.0
	4	0	0	0.2	0.6	0.6	0.6	1.0

(d) RSP to DP

		DP						
		-3	-2	-1	0	1	2	3
RSP	0	0.2	0.4	0.4	0.6	0.6	0.8	0.6
	1	0.4	0.6	0.6	0.6	0.8	1.0	0.6
	2	0.6	0.6	0.8	0.2	1.0	0.8	0.6
	3	0.6	0.8	1.0	0.6	0.8	0.6	0.6
	4	0.6	1.0	0.8	0.6	0.6	0.6	0.4

(e) RSP to DD

		DD						
		-3	-2	-1	0	1	2	3
RSP	0	0	0	0.2	0.6	0.8	0.6	0.2
	1	0	0	0.2	0.6	1.0	0.6	0.2
	2	0	0	0.2	0.6	1.0	0.6	0.2
	3	0	0	0.2	0.6	1.0	0.6	0.2
	4	0	0	0.2	0.6	0.8	0.6	0.2

**Table 3.3:** (cont'd)

(f) DIV to DP

		DP							
		-3	-2	-1	0	1	2	3	
DIV		0	0.2	0.6	0.8	0.6	0.2	0	0
	1	0.4	0.6	1.0	0.6	0.2	0	0	0
	2	0.6	0.6	1.0	0.6	0.2	0	0	0
	3	0.6	0.8	1.0	0.6	0.2	0	0	0
	4	0.6	1.0	0.8	0.6	0.2	0	0	0

(g) DIV to DI

		DI							
		-3	-2	-1	0	1	2	3	
DIV		0	0.2	0.4	0.4	0.4	0.2	0	0
	1	0.4	0.6	0.6	0.6	0.2	0	0	0
	2	0.6	0.6	0.8	0.6	0.2	0	0	0
	3	0.6	0.8	1.0	0.6	0.2	0	0	0
	4	0.6	1.0	0.8	0.6	0.2	0	0	0

(h) DIV to DD

		DD							
		-3	-2	-1	0	1	2	3	
DIV		0	0	0	0.2	0.6	0.8	0.6	0.6
	1	0	0	0.2	0.6	1.0	0.8	0.6	0.6
	2	0	0	0.2	0.6	0.8	1.0	0.8	0.8
	3	0	0	0.2	0.6	0.6	0.8	1.0	1.0
	4	0	0	0.2	0.4	0.6	0.6	1.0	1.0

(i) OFS to DP

		DP							
		-3	-2	-1	0	1	2	3	
OFS		0	0	0	0.2	0.6	0.8	0.6	0.2
	1	0	0	0.2	0.6	1.0	0.6	0.2	0.2
	2	0	0	0.2	0.6	0.8	0.6	0.2	0.6
	3	0	0	0.2	0.6	1.0	0.8	0.6	0.6
	4	0	0	0.2	0.6	0.8	1.0	0.6	0.6

(j) OFS to DI

		DI							
		-3	-2	-1	0	1	2	3	
OFS		0	0	0	0.2	0.6	0.8	0.6	0.6
	1	0	0	0.2	0.6	1.0	0.8	0.6	0.6
	2	0	0	0.2	0.6	0.8	1.0	0.8	0.8
	3	0	0	0.2	0.6	0.6	0.8	1.0	1.0
	4	0	0	0.2	0.4	0.6	0.6	1.0	1.0

**Table 3.4:** Fuzzy decision table for a servo controller

<b>Condition</b>	<b>Action</b>		
	<b>DP</b>	<b>DI</b>	<b>DD</b>
NON	0.0	0.0	0.0
OSC = LOW	-1.412	-0.314	0.686
= MOD	-0.714	-0.343	0.714
= HIG	-0.743	-0.343	0.743
= VHI	-0.714	-0.371	0.714
RSP = LOW	0.114	0.0	0.543
= MOD	0.029	0.0	0.543
= HIG	-0.029	0.0	0.543
= VHI	-0.114	0.0	0.543
DIV = LOW	-0.543	-0.514	0.686
= MOD	-0.600	-0.571	0.714
= HIG	-0.600	-0.600	0.743
= VHI	-0.629	-0.629	0.713
OFS = LOW	0.543	0.657	0.0
= MOD	0.571	0.714	0.0
= HIG	0.600	0.743	0.0
= VHI	0.629	0.714	0.0

matched with the rule base obtained in Table 3.3. This is accomplished by applying the compositional rule of inference, as given by equation (3.4). Recall that this is a “sup of min” operation. Specifically, we compare the membership function vector of a response condition with each column of a relation table (Table 3.3), take the lower value in each pair of compared elements and then take the largest of the vector elements thus obtained. This result now has to be defuzzified, in order to obtain a crisp value for the tuning action. The centroid method (equation (3.13)) is used here. Specifically, we weight the elements in the universe (strictly, the support set) of the action variable using the membership grades of the action, and then take the average by dividing the sum of these membership grades. The fuzzy decision table obtained by following these steps for every rule in the fuzzy rule base is given in Table 3.4. It has been decided to take no action when the conditions are satisfactory, even though an optimal tuning strategy would suggest some other action under these conditions. The relation used for updating a PID parameter is

$$p_{\text{new}} = p_{\text{old}} + \Delta p(p_{\text{max}} - p_{\text{min}})/p_{\text{sen}} \quad (3.45)$$

in which  $p$  denotes a PID parameter. The subscript “new” denotes the updated value and “old” denotes the previous value. The incremental action taken by the fuzzy controller is denoted by  $\Delta p$ . Upper and lower bounds for a parameter are denoted by the subscripts “max” and “min”. A sensitivity parameter  $p_{\text{sen}}$  is also introduced for adjusting the sensitivity of tuning, when needed.

When using the decision table given in Table 3.4, it is assumed that an intelligent preprocessor is available to monitor the process behavior and

determine the condition of the process attributes (OSC, RSP, DIV, and OFS) as fuzzy states rather than numerical values. In fact, a measured numerical value may be fuzzified to establish the fuzzy context for table look-up. Alternatively, it is also possible to construct a decision table where the conditions are expressed as discrete numerical values. Then, either some form of interpolation or rounding-off to the nearest discrete value would be necessary. For example, an interval of numerical values may be assigned to each fuzzy state, and if the measured value falls within this interval, the particular condition variable is taken to have this fuzzy state, for the purpose of table look-up.

### 3.5 Properties of fuzzy control

Several ideal requirements and analytical properties of a fuzzy controller are presented in this section. These considerations are important in the design and development of a fuzzy controller. The ideal requirements provide a reference point for evaluating a practical fuzzy controller.

#### 3.5.1 Fuzzy controller requirements

In fuzzy logic inference, and particularly in fuzzy logic control, one may run into difficulties unless certain conditions are satisfied by the associated knowledge base. Specifically, the following ideal conditions may be specified:

- (1) The rule base should be “complete.”
- (2) The rule base should be “continuous.”
- (3) The rules should be “consistent.”
- (4) The rules should not “interact.”
- (5) The rule based system should be “robust” and “stable.”

It should be emphasized that these are rather ideal requirements, some being contradictory and others complementary. Now, let us indicate the meaning of each of these requirements. Specifically, consider a fuzzy logic rule base  $R$  with the condition fuzzy variables (i.e., context or antecedent fuzzy sets)  $X$  and the action fuzzy variables (i.e., consequent fuzzy sets)  $C$ , such that,

$$R: \bigcup_i X_i \rightarrow C_i \quad (3.46)$$

where the operator  $\cup$  denotes an OR (or union) operation, which usually corresponds to a “max” operation on the membership functions. Now for a particular context (or response measurement or observation)  $\hat{X}_i$ , the corresponding fuzzy inference  $\hat{C}_i$  is determined by applying the *composition operation*, as

$$\hat{X}_i \circ R = \hat{C}_i \quad (3.47)$$

This notation is used in the following discussion.

### 3.5.2 Completeness

A rule base is said to be complete if it provides a meaningful inference (control action) for any possible context (i.e., for any possible observation or system response). In other words, at least one rule has to be present in the rule base for every possible condition of the system. More specifically, for any possible observation (i.e., fuzzy measurement or data)  $\hat{X}$  in the observation space  $X$  and the corresponding inference  $\hat{C}$  (fuzzy action) there should be at least one corresponding rule  $i: (X_i \rightarrow C_i)$  in the rule base such that  $\hat{X}$  is representable by  $X_i$  and  $C_i$  is representable by  $C_i$ . Analytically, we may express this requirement as follows.

A rule base is complete if for all possible observations (i.e., fuzzy measurement or data)  $\hat{X}$  and the corresponding fuzzy inferences  $\hat{C} = \hat{X} \circ R$ , there exists a rule  $i$  in the rule base such that the rule condition  $X_i$  has some overlap with  $\hat{X}$ , and the rule action  $C_i$  has some overlap with  $\hat{C}$ . In mathematical notation we write:

$$\forall_{\hat{X} \in X} \exists X_i \rightarrow C_i \text{ such that } \text{hgt}(\hat{C} \cap C_i) > 0 \text{ and } \text{hgt}(\hat{X} \cap X_i) > 0 \quad (3.48)$$

Here  $\text{hgt}(A)$  is the “height of the membership function” of  $A$ , as defined by  $\text{hgt}(A) = \sup_a \mu_A(a)$ , which is simply the global peak value of the membership function.

The completeness of a rule base assures that a satisfactory control inference is made for any valid set of data (observations). Rule base completeness need not be strictly satisfied for a rule-based system to operate satisfactorily. Often, less important or unknown rules are omitted from a rule base without seriously affecting the system performance. In the matrix representation of a rule base, the empty grid elements are an indication of an incomplete rule base.

### 3.5.3 Continuity

A rule base is said to be continuous if there are no “gaps” between any rules, with respect to their membership functions. Specifically, when the condition fuzzy sets overlap, the action fuzzy sets also should overlap. To analytically express this condition, first let us define the *possibility function* of two fuzzy sets  $A$  and  $B$  as the membership function of the set intersection  $A \cap B$ . Specifically, the possibility function

$$F_{A,B}(x) = \mu_{A \cap B}(x), \quad x \in X \quad (3.49)$$

where  $X$  is the universe of discourse of both  $A$  and  $B$ . Then, a rule base is continuous if for some rule  $i: X_i \rightarrow C_i$

there exists at least one other rule  $j \neq i$ :  $X_j \rightarrow C_j$

with non-zero condition possibility  $F_{X_i, X_j}(x)$  for some  $x$  (i.e.,  $X_i \cap X_j \neq \emptyset$ ) and non-zero action possibility  $F_{C_i, C_j}(c)$  for the some  $c$  (i.e.,  $C_i \cap C_j \neq \emptyset$ ).

The degree of overlap of the condition variables and action variables of a rule base has a bearing on the nature of continuity of the rule base. A continuous control rule base with proper levels of overlap in the adjacent rules can help provide smooth and chatter-free performance in the control system. As in the case of *centroid defuzzification*, a continuous rule base may result in a slow (sluggish) performance, however.

### 3.5.4 Consistency

A rule base is said to be consistent when there are no contradictory rules. Three separate situations need to be addressed.

#### *Case 1*

Suppose that a rule base has the following two rules:

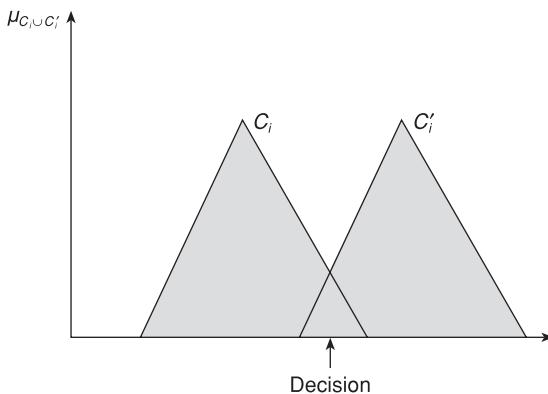
$$\begin{aligned} & X_i \rightarrow C_i \\ & \text{Or} \\ & X_i \rightarrow C'_i \end{aligned} \tag{3.50}$$

where for the same context (antecedent)  $X_i$  two consequents  $C_i$  and  $C'_i$  in the *same universe* are possible. These two rules are not necessarily contradictory (inconsistent). If these consequents are mutually exclusive (disjoint), i.e.,  $C_i \cap C'_i = \emptyset$ , however, the rule base is said to be *inconsistent* (or contradictory).

If a rule base  $R$  is inconsistent, the fuzzy inference  $\hat{C}$  for a fuzzy measurement  $\hat{X}$ , which is obtained through the composition  $\hat{C} = \hat{X} \circ R$ , can be *multi-modal*. That means the membership function  $\mu_{\hat{C}}(c)$  of the inference can have more than one peak. This is likely the case when there are two or more contradictory rules that are very strong. To explore this further, note that the two rules in (3.50) correspond to the single rule

$$X_i \rightarrow C_i \cup C'_i \tag{3.50}^*$$

Since  $C_i$  and  $C'_i$  are in the same universe, the membership function of the overall consequent may be represented as in Figure 3.18. Note that if the centroid method of defuzzification is used, the crisp inference (decision) is at the centroid of the shaded area, as shown. This point is close to neither of the two modes (peaks) of the consequent membership function. It represents a weak compromise, and is by no means an effective inference. The most “inconsistent” situation arises when  $C_i$  and  $C'_i$  are disjoint; i.e.,  $C_i \cap C'_i = \emptyset$ . In this case, the associated inferences are of no great value. As the degree of overlap between  $C_i$  and  $C'_i$  increases, the consistency increases (and the degree of



**Figure 3.18:** Consistency of two rules connected by OR

contradiction decreases). The most consistent situation corresponds to  $C_i = C'_i$ , in which case there is no contradiction in the two rules (3.50). A measure of consistency would be  $\text{hgt}(C_i \cap C'_i)$ .

### Case 2

Now suppose that a rule base has the following two rules:

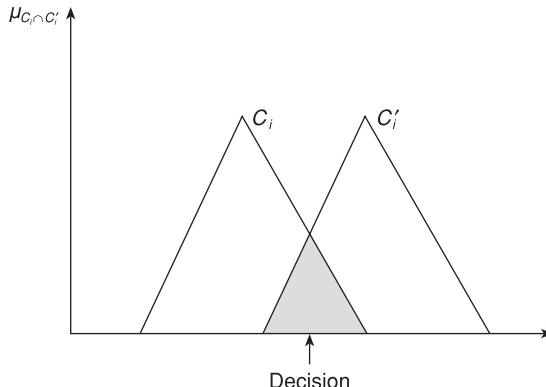
$$\begin{aligned} X_i &\rightarrow C_i \\ \text{And} \\ X_i &\rightarrow C'_i \end{aligned} \tag{3.51}$$

where, as before, the two consequents  $C_i$  and  $C'_i$  are in the *same universe*. Again, the two rules are not necessarily inconsistent. If these consequents are mutually exclusive (disjoint), i.e.,  $C_i \cap C'_i = \emptyset$ , then the rule base is said to be *inconsistent*.

To explore this case further, note that the two rules in (3.51) correspond to the single rule

$$X_i \rightarrow C_i \cap C'_i \tag{3.51}^*$$

Since  $C_i$  and  $C'_i$  are in the same universe, the membership function of the overall consequent may be represented as in Figure 3.19. Note that if the centroid method of defuzzification is used, the crisp inference (decision) is at the centroid of the shaded area, as shown. This point being close to neither of the two modes (peaks) of the consequent membership function, it represents a weak compromise as for the previous case. The most inconsistent (contradictory) situation arises when  $C_i$  and  $C'_i$  are disjoint, i.e.,  $C_i \cap C'_i = \emptyset$ . In this case, (3.51)\* represents a “no-action” rule, and is of no particular value. As the degree of overlap between  $C_i$  and  $C'_i$  increases, the consistency increases (and the degree of contradiction decreases). The most consistent situation



**Figure 3.19:** Consistency of two rules connected by AND

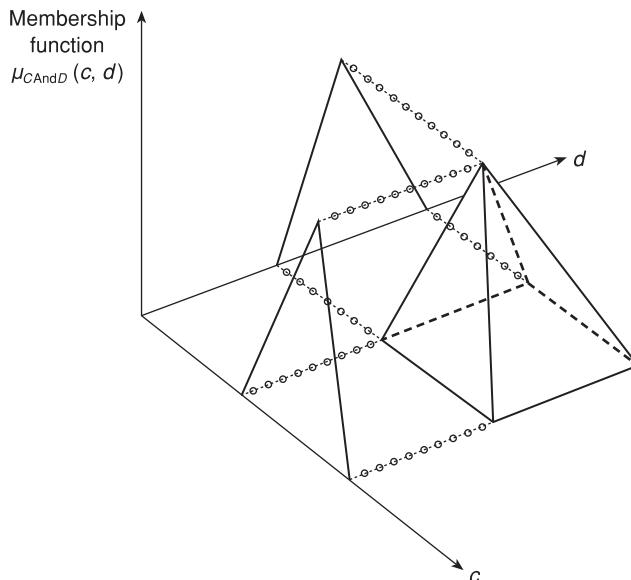
corresponds to  $C_i = C'_i$ , in which case there is no contradiction in the two rules (3.51). A measure of consistency would be  $\text{hgt}(C_i \cap C'_i)$ .

### Case 3

Suppose that a rule base has the following rule:

$$X_i \rightarrow C \text{ And } D \quad (3.52)$$

where the two consequents  $C$  and  $D$  are in *different universes*. This rule is schematically shown in Figure 3.20. Here the consequent is determined by



**Figure 3.20:** Rule consistency when the two consequents are in different universes

the membership functions of the component consequents, which are independent. Hence, this case does not represent an inconsistency since there is no contradiction.

### 3.5.5 Rule validity

A rule in a knowledge base becomes a fuzzy statement when the *rule-antecedent* (context) and the *rule-consequent* (action) are fuzzy descriptors represented by fuzzy sets. Apart from the fuzziness of these two types of descriptors, the validity of the rule itself may be questioned. A level of validity may be assigned to the rule depending on such factors as the amount of past experience, evidence, and overall knowledge about the rule. The level of validity may be modified as more knowledge is gained through *learning* and other means.

Consider the fuzzy logic rule: IF  $A$  THEN  $B$ . The decision based on this rule may be affected by several factors. For example:

- (1) If the perception of either of the fuzzy variables  $A$  and  $B$  changes, then the associated membership function ( $\mu_A$  or  $\mu_B$ ) will change and as a result the decision based on the rule may change.
- (2) If the data (observations, measurements, etc.) that determine the context of  $A$  change as a result of a change in state of the fuzzy logic system, the corresponding grade of membership of the context will change (in the fuzzification operation) and consequently the decision based on the rule may change even when the perception of the fuzzy variables (membership functions of)  $A$  and  $B$  have not changed.
- (3) If the level of belief of the rule itself changes as a result of learning and further knowledge, then the decision based on the rule may change even though the membership functions of  $A$  and  $B$  and the context data for  $A$  have not changed.

The consideration of item 3 above may be incorporated into a rule by introducing a *validity factor*  $b$  as: IF  $A$  THEN  $B$  WITH Belief  $b$ , where  $b \in [0, 1]$ . Here, the processing of the rule may proceed as usual (using individual rule-based composition), but the resulting fuzzy-inference (membership function) has to be scaled by the factor  $b$  prior to combining (aggregating) the inferences from the individual rules and subsequent defuzzification. The result will be identical to scaling by  $b$  the membership function of  $B$ , but the underlying concepts (the membership function of  $B$  and the belief level of the rule) are quite different. Furthermore, the parameter  $b$  in the rule given above may represent any compatible concept such as “confidence,” “certainty,” “belief,” “plausibility,” “validity,” and “feasibility” of the rule.

### 3.5.6 Rule interaction

There exists rule interaction if the inference from a particular rule is affected by other rules in the rule base. Rule interaction is not necessarily an undesirable

thing even though a non-interacting rule base leads to simpler and faster decision-making. Rule interaction can be addressed from two points of view, as indicated below.

### *Condition 1*

Consider a rule base  $R$ . We say that there exists rule interaction, if there exists ( $\exists$ ) some rule  $i: X_i \rightarrow C_i$  such that

$$X_i \circ R \neq C_i \quad (3.53)$$

In other words, if the rules in the rule base interact, then the *antecedent* (context)  $X_i$  of a particular rule, when composed with the rule base  $R$ , may not necessarily result in the *consequent*  $C_i$  as given by the rule itself. This is a sufficient condition for rule interaction, but not a necessary condition.

Some care should be exercised in interpreting this condition. If the rule base  $R$  is formed by just one rule, then if the antecedent  $X$  of this rule is composed with  $R$  the result will be  $C$ , which is the consequent of the original rule itself. Of course, this assumes that consistent logical operations are used both in the formation of  $R$  and in the application of the compositional rule (for example, *min* operation for implication and *sup-min* for composition). This should be intuitively clear because there cannot be rule interaction unless there is more than one rule. When there are several rules, however, the stronger rules will tend to dominate, and rule interaction may be present. Then the condition (context) of a weak rule, when composed with the overall rule base, may not provide exactly the action (consequent) as predicated by the rule itself. As indicated before, it should be clear that such interaction is not necessarily a bad thing because a weaker rule would be an uncertain or inaccurate rule.

### *Condition 2*

Suppose that a rule base has the following two rules:

$$X_i \rightarrow C_i$$

Or

$$X_j \rightarrow C_j$$

where for the context variables (antecedents)  $X_i$  and  $X_j$  are in the same universe, and the two consequents  $C_i$  and  $C_j$  are also in the same universe, such that  $X_i \cap X_j = \emptyset$  (i.e., disjoint). Then, if  $C_i \cap C_j \neq \emptyset$ , the rules are said to interact. This is a very stringent (strong) condition. Instead, we may use  $X_i \cap X_j = \phi^*$  and  $C_i \cap C_j \neq \phi^{**}$ , where  $\phi^*$  and  $\phi^{**}$  are fuzzy null sets.

### 3.5.7 Rule base decoupling

The computational and developmental advantages of using an uncoupled rule base in fuzzy control are tremendous. Generally, however, some accuracy is

lost by assuming that the rules in a knowledge base are uncoupled. In view of this, it is important to examine the conditions under which this assumption can be made without sacrificing the control accuracy. As an example, consider a fuzzy logic controller with the coupled rule base:

```
If  $A_1$  and  $B_1$  then  $C_1$  and  $D_1$ 
else if  $A_2$  and  $B_2$  then  $C_2$  and  $D_2$ 
end if. . . .
```

We wish to investigate the conditions under which, without loss of accuracy, this coupled rule base may be expressed as

```
If  $A_1$  then  $C_1^*$  and  $D_1^*$ 
If  $B_1$  then  $\bar{C}_1$  and  $\bar{D}_1$ 
If  $A_2$  then  $C_2^*$  and  $D_2^*$ 
If  $B_2$  then  $\bar{C}_2$  and  $\bar{D}_2$ 
```

Now a theoretical basis of rule base decoupling is presented. First, the general fuzzy control problem with a coupled rule base is formulated. This represents a multi-degree-of-freedom decision-making problem. Next, the method of single degree-of-freedom decision-making using a coupled rule base is given. Finally, the assumption of uncoupled rule base is incorporated into the single-degree-of-freedom decision-making problem. On that basis, conditions are established for rule base decoupling in the problem of single degree-of-freedom decision-making.

### 3.5.7.1 Decision-making through a coupled rule base

Consider a knowledge base of fuzzy control, given as a set of linguistic rules in the general form

$$\text{Else}[ \text{If } e_1 \text{ is } E_1^i \text{ and } \dots \text{ and } e_n \text{ is } E_n^a \text{ then } c_1 \text{ is } C_1^i \text{ and } \dots \text{ and } c_p \text{ is } C_p^b ] \quad (3.54)$$

with the condition vector  $e\Delta[e_1, \dots, e_n]^T \in \Re^n$  and the action vector  $c\Delta[c_1, \dots, c_p]^T \in \Re^p$ . Suppose that by monitoring the condition vector  $e$  of the system, a fuzzy context  $\hat{E}\Delta[\hat{E}_1, \hat{E}_2, \dots, \hat{E}_n]^T$  has been established at a given instant. This will provide the membership function  $\mu_{\hat{E}}(e)$  of the context. The control decision  $\hat{C}\Delta[\hat{C}_1, \hat{C}_2, \dots, \hat{C}_p]^T$  corresponding to the context  $\hat{E}$  is computed through the use of the compositional rule of inference, as described previously in this chapter. Specifically, we get

$$\mu_{\hat{C}}(c) = \sup_e \min [\mu_{\hat{E}}(c), \mu_R(e, c)] \quad (3.55)$$

Then, crisp control actions  $\hat{c}$  have to be computed from fuzzy  $\hat{C}$ . In this process, first the action membership function given by equation (3.55) is

projected along each axis  $c_q$  and then the centroid method is applied to compute the corresponding control action.

**Definition:** Consider a membership function  $\mu(x, y): \mathbb{R}^n \times \mathbb{R}^p \rightarrow [0, 1]$ . Its *projection* in the  $X_i \times Y_j$  subspace is denoted by  $\text{Proj}[\mu(x, y)](x_i, y_j): \mathbb{R} \times \mathbb{R} \rightarrow [0, 1]$  and is given by

$$\text{Proj}[\mu(x, y)](x_i, y_j) \Delta \sup_{\substack{\forall x_k \neq x_i \\ \forall y_l \neq y_j}} \mu(x, y) \quad (3.56)$$

Using this definition of projection, the crisp control actions corresponding to equation (3.55) are computed by

$$\hat{c}_q = \frac{\int c_q \text{Proj}[\mu_C(c)](c_q) dc_q}{\int \text{Proj}[\mu_C(c)](c_q) dc_q} \quad (3.57)$$

$$q = 1, 2, \dots, p$$

in which each integration is performed over the support set of the particular membership function.

The use of pairs of a single condition variable and a single action variable is quite realistic in fuzzy logic control. Now let us address the decision-making problem from this particular point of view. Consider a coupled rule base  $R$  as given by equation (3.54), with membership function  $\mu_R(e, c)$ . Suppose that a crisp observation  $\hat{e}_s$  of the condition variable  $e_s$  is available, and it is required to determine a crisp control value  $\hat{c}_q$  for the corresponding action variable  $c_q$ . First the projection of  $\mu_R$  in the  $E_s \times C_q$  subspace is determined as  $\text{Proj}[\mu_R(e, c)](\hat{e}_s, c_q)$  using equation (3.56). The crisp condition  $\hat{e}_s$  may be expressed as a fuzzy singleton whose membership function is

$$\begin{aligned} \mu_{\hat{E}_s}(e) \Delta s(e - \hat{e}_s) &= 1 \quad \text{for } e = \hat{e}_s \\ &= 0 \quad \text{elsewhere} \end{aligned} \quad (3.58)$$

Then, by applying the compositional rule of inference (3.55) and the centroid equation (3.57), it can be shown that

$$\hat{c}_q = \frac{\int c_q \text{Proj}[\mu_R(e, c)](\hat{e}_s, c_q) dc_q}{\int \text{Proj}[\mu_R(e, c)](\hat{e}_s, c_q) dc_q} \quad (3.59)$$

$$q = 1, 2, \dots, p$$

$$s = 1, 2, \dots, n$$

This result is used in making a control decision in the case of a coupled rule base.

### 3.5.7.2 Decision-making through an uncoupled rule base

Now the rule base is assumed to be uncoupled, and subgroups of rules are considered separately, where each subgroup relates just one condition variable  $e_s$  and one action variable  $c_q$ ; thus

$$R_{s,q} : \text{Else}_{i,j(i)} [\text{If } e_s \text{ is } E_s^i \text{ then } c_q \text{ is } C_q^j] \quad (3.60)$$

Note that the membership function of the rule base subgroup is given by

$$\mu_{R_{s,q}}(e, c) = \max_{i,j(i)} \min[\mu_{E_s^i}(e) \mu_{C_q^j}(c)] \quad (3.61)$$

For a fuzzy context  $\hat{E}_s$  that is given at a particular instant, the corresponding fuzzy action  $\hat{C}_q$  is obtained by applying the compositional rule of inference in the usual manner, as

$$\mu_{\hat{C}_q}(c) = \sup_e \min[\mu_{\hat{E}_s}(e), \mu_{R_{s,q}}(e, c)] \quad (3.62)$$

If the context is assumed to be a fuzzy singleton of value  $e_s$ , the corresponding crisp action is determined by the centroid method; thus

$$\hat{c}_q = \frac{\int c \mu_{R_{s,q}}(\hat{e}_s, c) dc}{\int \mu_{R_{s,q}}(\hat{e}_s, c) dc} \quad (3.63)$$

Even though the computational requirements of decision-making can be significantly reduced by the assumption of an uncoupled rule base, some accuracy is lost here. This will be addressed next. The concepts of a coupled rule base can be directly applied to an uncoupled rule base, through the use of the familiar concept of cylindrical extension, which is defined now.

**Definition:** Consider  $x \Delta [x_1, x_2, \dots, x_n]^T \in \Re^n$ ,  $y \Delta [y_1, y_2, \dots, y_p]^T \in \Re^p$ , and  $\mu(x_i, y): \Re \times \Re^p \rightarrow [0, 1]$  with  $1 \leq i \leq n$ . The *cylindrical extension* of  $\mu(x_i, y)$  over the entire space  $\Re^n \times \Re^p$  is given by

$$\begin{aligned} cyl[\mu(x_i, y)](x, y) &: \Re^n \times \Re^p \rightarrow [0, 1] \\ &= \mu(x_i, y), \quad \forall x, y; 1 \leq i \leq n \end{aligned} \quad (3.64)$$

With this definition, it is clear that the membership function of the entire rule base is the *intersection* of the cylindrical extensions of the membership functions of the subgroups; thus

$$\mu_R(e, c) = \Lambda_{s,q(s)} cyl[\mu_{R_{s,q}}(e_s, c_q)](e, c) \quad (3.65)$$

The uncoupled rule base will not suffer any further loss of accuracy through this extension in view of the fact that, in the uncoupled case, we have

$$\text{Proj}[cyl[\mu_{R_{s,q}}(e_s, c_q)](e, c)](e_s, c_q) = \mu_{R_{s,q}}(e_s, c_q) \quad (3.66)$$

### 3.5.7.3 Equivalence condition

From the developments presented thus far, it is clear that the control inferences made using a coupled rule base are generally not the same as those made by assuming an uncoupled rule base, for the same system parameters and data. But it can be shown that if a certain condition is satisfied by the membership functions of the rule base variables, the control inferences in the two cases become identical, with an obvious computational advantage in the case of the uncoupled rule base. A condition for this equivalence is established now.

Zadeh developed the concept of *separability* of fuzzy restrictions, and showed that a fuzzy restriction is separable if and only if the *join* of the projections of the restrictions results in the original restriction. Note that the join is the intersection of the cylindrical extensions. This concept can be extended to establish the equivalence condition mentioned above. Again consider the coupled rule base given by (3.54). Its membership function may be expressed as

$$\begin{aligned} \mu_R(e, c) = \max_{i,j(i)} \min & [\mu_{E_1^{i(1)}}(e_1), \mu_{E_2^{i(2)}}(e_2), \dots, \mu_{E_n^{i(n)}}(e_n), \\ & \mu_{C_1^{j(1)}}(c_1), \mu_{C_2^{j(2)}}(c_2), \dots, \mu_{C_p^{j(p)}}(c_p)] \end{aligned} \quad (3.67)$$

where  $i$  denotes a coupled rule in the rule base, and hence represents the associated set of fuzzy states of the condition variables, and  $j$  denotes the associated set of fuzzy states of the action variables. The *min* operation of the membership functions in equation (3.67) may be interpreted as the intersection of their cylindrical extensions in  $\Re^n \times \Re^p$ . Then, when using equation (3.67) in (3.59) for making inferences through a coupled rule base, it is required to determine the projection  $\Re^n \times \Re^p \rightarrow \Re \times \Re$  given by

$$\text{Proj}[\mu_R(e, c)](e_s, c_q) = \sup_{\substack{\forall e_k \neq e_s \\ \forall c_l \neq c_q}} \{\mu_R(e, c)\} \quad (3.68)$$

Now, by substituting equation (3.67) into (3.68) it can be shown that

$$\text{Proj}[\mu_R(e, c)](e_s, c_q) = \max_{i,j(i)} \min [\mu_{E_s^{i(s)}}(e_s), \mu_{C_q^{j(q)}}(c_q), \mu_i] \quad (3.69)$$

in which

$$\mu_i = \min_{k \neq s} \sup_{e_k, c_\ell} [\mu_{E_k^{i(k)}}(e_k), \mu_{C_\ell^{j(\ell)}}(c_\ell)] \quad (3.70)$$

for the  $i$ -th rule.

Then by comparing equation (3.69) with (3.61) it follows that the equivalence condition is

$$\mu_i \geq \min[\sup_{e_s, c_q} \mu_{E_s^{i(s)}}(e_s), \mu_{c_q^{k(q)}}(c_q)] \quad (3.71)$$

A special case in which the condition (3.71) is satisfied is when all the fuzzy variables in the rule base are *normal* (i.e., when they have peak membership grades of unity). In this case  $\mu_i = 1$  for all  $i$ .

## 3.6 Robustness and stability

Robustness of a control system denotes the insensitivity to parameter changes, model errors (in model-based control), and disturbances (including disturbance inputs and noise in various control signals). Stability of a control system refers to a bounded response when the inputs themselves are bounded. This is termed *bounded-input-bounded-output* (BIBO) stability. A special case is *asymptotic stability* where the response asymptotically approaches zero (origin) when excited from the origin and the inputs are maintained at zero value thereafter. Then, robustness may be interpreted as stability under system disturbances.

The degree of stability of a control system is a measure of distance to the state of *marginal stability* (i.e., almost unstable or steadily oscillating state under zero input conditions). This distance is termed the *stability margin* of which *phase margin* and *gain margin* are special measures for linear systems in the frequency domain. Similarly, a *robustness index* of a control system may be determined by establishing a representative bound for a system parameter or signal disturbance within which the control system will remain stable but outside which it is likely to become unstable. Note, however, that fuzzy logic control is not a model-based technique in the sense that it does not employ an explicit model of the process. Consequently, in this case, robustness cannot be defined with respect to model errors. Since robustness is traditionally defined in terms of stability, we shall explore the latter topic further, from two different points of view. First we need to define a fuzzy dynamic system.

### 3.6.1 Fuzzy dynamic systems

Consider a nonlinear and time-invariant dynamic system expressed in the discrete state-space form:

$$x_{n+1} = f(x_n, u_n) \quad (3.72)$$

$$y_n = g(x_n) \quad (3.73)$$

For the purpose of explaining the underlying concepts, only the scalar case is considered. The state variable  $x$  and the output variable  $y$  are assumed

to be fuzzy, and the input variable  $u$  is assumed to be crisp for the time being. Generally, in a fuzzy system the state transition relation  $f$  and the output relation  $g$  both will be fuzzy relations with fuzzy sets  $F$  and  $G$ . Suppose that the fuzzy sets corresponding to  $x_n$ ,  $x_{n+1}$ , and  $y_n$  are  $X(n)$ ,  $X(n + 1)$ , and  $Y(n)$ , respectively. A typical simulation objective for a fuzzy dynamic system would be to determine the membership functions of  $X(n + 1)$  and  $Y(n)$  assuming that the membership functions of  $X(n)$ ,  $F$ ,  $G$  are known. The “composition” operation, not the extension principle, is applicable here, because the relations  $f$  and  $g$  are also fuzzy. Specially, we have the following results:

$$\mu_{X(n+1)}(x_{n+1}) = \sup_{x_n \in X} \min\{(\mu_{X(n)}(x_n), \mu_F(x_{n+1}, x_n, u_n))\} \quad (3.74)$$

$$\mu_{Y(n)}(y_n) = \sup_{x_n \in X} \min\{(\mu_{X(n)}(x_n), \mu_G(x_n, u_n))\} \quad (3.75)$$

Note that  $X$  denotes the universe in which the state variable  $x$  lies; in this case the state space. It should be clear that the “composition” operation is used in the two results (3.74) and (3.75). Specifically, the fuzzy set  $X(n + 1)$  is obtained through “matching” the fuzzy set  $X(n)$  with the fuzzy relation  $F$  and projecting the result into the inference subspace. Similarly, the fuzzy set  $Y(n)$  is obtained through composition of the fuzzy set  $X(n)$  and the fuzzy relation  $G$ . In equation (3.74), for example, the operation  $\min$  is applied for the membership functions of  $X(n)$  and  $F$  because the element  $x_n$  is a variable of the relation  $f$  and the two are matched through an AND operation. The operation  $\sup$  is applied over the universe of  $X(n)$  because all the elements  $x_n$  in the membership function of  $X(n)$  are mapped to the same element  $x_{n+1}$  in the membership function of  $X(n + 1)$  and hence, the most desirable (possible) mapping must be chosen. This is exactly the idea of composition.

It is straightforward to extend the relations (3.74) and (3.75) to the case where the input  $u_n$  is also fuzzy. Then, the membership function of the corresponding fuzzy set  $U(n)$  should be included as well in the  $\min$  operation, and furthermore, the  $\sup$  operation should be carried out over the universe of  $U(n)$  as well, giving an added dimension (input) to the composition.

### 3.6.2 Stability of fuzzy systems

Two approaches may be used to study stability in a fuzzy system:

- (1) Represent the fuzzy controller by a nonlinear model, through simplifying assumptions, and perform stability analysis using traditional approaches (e.g., Lyapunov function method).
- (2) Interpret stability as the stability of the fuzzy logic inference mechanism. Study stability of the decision-making system, largely at the system level.

#### 3.6.2.1 Traditional approach to stability analysis

The first approach is what is predominantly used in the published literature, perhaps due to the convenience and availability of well-established techniques,

particularly Lyapunov-like approaches for the stability analysis of nonlinear systems. However, this is a somewhat artificial (or synthetic) way of dealing with stability of a fuzzy system, and may not be generally valid. The reason is simple. Once the fuzzy subsystem is represented by an analytic nonlinear model with nonfuzzy parameters and variables, one no longer has a fuzzy system with its inherent fuzzy features. Specifically, consider the nonlinear state space model given by

$$\frac{d\mathbf{X}}{dt} = \mathbf{f}(\mathbf{x}, \mathbf{u}) \quad (3.76)$$

$$\mathbf{y} = \mathbf{g}(\mathbf{x}, \mathbf{u}) \quad (3.77)$$

where  $\mathbf{x} \in \Re^n$  is the state vector,  $\mathbf{y} \in \Re^m$  is the output (system response) vector, and  $\mathbf{u} \in \Re^r$  is the input (control action) vector. Suppose that a fuzzy logic rule base  $R$  is available that relates the fuzzy response variables  $\mathbf{Y}$  to fuzzy control action variables  $\mathbf{U}$ :

$$R: \mathbf{Y} \rightarrow \mathbf{U} \quad (3.78)$$

A crisp response  $\mathbf{y}$  has to be first fuzzified using an  $FZ$  operator (say, reading off the membership grades from the membership functions of  $\mathbf{Y}$ ):

$$\hat{\mathbf{Y}} = FZ(\mathbf{y}) \quad (3.79)$$

Next, apply the compositional rule of inference to obtain a fuzzy control action  $\hat{\mathbf{U}}$ :

$$\hat{\mathbf{U}} = \hat{\mathbf{Y}} \circ R \quad (3.80)$$

and finally, defuzzify using the  $FZ^{-1}$  operator (say, employing the *centroid method*) as:

$$\mathbf{u} = FZ^{-1}(\hat{\mathbf{U}}) \quad (3.81)$$

The overall nonlinear controller that is equivalent to the fuzzy controller is then

$$\mathbf{u} = h(\mathbf{y}) \quad (3.82)$$

where

$$\mathbf{b} = FZ^{-1}(FZ(\mathbf{y}) \circ R) \quad (3.83)$$

Note that even though the nonlinear controller given by equation (3.81) looks rather innocent and may be employed in the traditional techniques of stability analysis, it is in fact an intractable function as given by

equation (3.83), unless greatly simplifying assumptions with regard to, for instance, vector dimensions, rules, and the membership functions are made. In particular, if a nonlinear control surface is established to represent a fuzzy controller, by following the approaches illustrated earlier (see Figure 3.11, for example), then Lyapunov's direct method (Lyapunov's 2nd method) may be applied to study the stability of that fuzzy controller.

### Example 3.4

Consider the single-link manipulator shown in Figure 3.21. The system parameters are:

$m$  = mass of the end effector

$l$  = length of the link

$k$  = torsional stiffness at the manipulator joint

$\tau$  = torque applied by the joint motor

$\theta$  = angle of rotation of the link, from the vertical position.

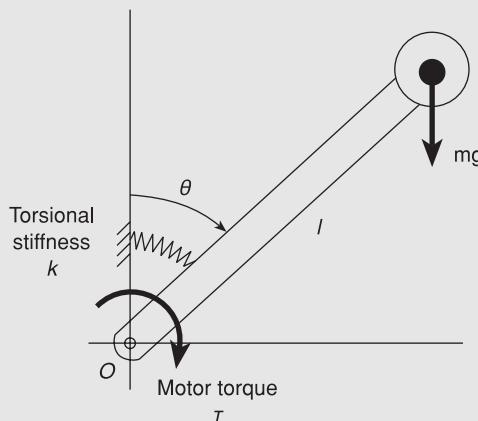


Figure 3.21: A single-link manipulator

By applying Newton's second law, the equation of angular motion of the manipulator is obtained as

$$ml^2\ddot{\theta} = lmg \sin \theta - k\theta + \tau \quad (i)$$

or

$$\ddot{\theta} = \frac{g}{l} \sin \theta - \frac{k}{ml^2} \theta + \frac{1}{ml^2} \tau$$

By defining the state variables:  $x_1 = \theta$ ,  $x_2 = \dot{\theta} = \dot{x}_1$ , we get the state-space model

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = p \sin x_1 - qx_1 + rt \quad (\text{ii})$$

where  $p = g/l$ ,  $q = k/(ml^2)$ , and  $r = 1/(ml^2)$ .

Suppose that the angular position ( $x_1$ ) and the angular speed ( $x_2$ ) are measured, and a fuzzy control action is generated on this basis, to drive the manipulator. Accordingly, the fuzzy control surface (developed in the usual manner) may be expressed by the nonlinear function

$$\tau = F(x_1, x_2) \quad (\text{iii})$$

Now let us try the Lyapunov function:

$$V(x_1, x_2) = \frac{1}{2}x_1^2 + \frac{1}{2}x_2^2 \quad (\text{iv})$$

which is positive definite.

For asymptotic stability of the system (by Lyapunov's direct method), we must have  $\dot{V} < 0$ . Now by differentiating (iv) with respect to time  $t$  and substituting (ii) and (iii), we have

$$\dot{V} = x_1 \dot{x}_1 + x_2 \dot{x}_2 = x_1 x_2 + x_2 [p \sin x_1 - qx_1 + rF(x_1, x_2)] < 0$$

for asymptotic stability.

This requires

$$F(x_1, x_2) < \frac{1}{r}[(q-1)x_1 - p \sin x_1] \quad (\text{v})$$

Once the “fuzzy” control surface  $F(x_1, x_2)$  is determined, we should check whether it satisfies the condition (v), by drawing the RHS function of (v). If the RHS function envelops the control surface  $F(x_1, x_2)$ , then the fuzzy control system is considered stable.

### 3.6.2.2 Composition approach to stability analysis

The second approach to the stability analysis of fuzzy systems concerns the stability of the fuzzy decision-maker itself. To illustrate this approach, consider a scalar fuzzy system  $F$  (i.e., a state equation) with crisp reference (fixed) input  $u$ , and expressed in the discrete-time form:

$$X_{n+1} = F(X_n, u) \quad (3.84)$$

where  $X_i$  is a fuzzy variable of system state. The system response is given by applying the compositional rule of inference, specifically the *sup-min* composition:

$$X_{n+1} = X_n \circ F \quad (3.85)$$

or

$$\mu_{X_{n+1}}(x_{n+1}) = \sup_{x_n \in X} \min \{\mu_{X_n}(x_n), \mu_F(x_{n+1}, x_n)\} \quad (3.86)$$

where the fuzzy system  $F$  is in fact a fuzzy rule base which relates  $X_i$  to  $X_{i+1}$ :

$$F: X_i \rightarrow X_{i+1} \quad (3.87)$$

Then, through successive application of the state transition relation, we get

$$X_{n+1} = X_1 \circ F^n \quad (3.88)$$

where  $F^n = F$  composed  $n - 1$  times.

The fuzzy system is considered *stable* if  $\lim_{n \rightarrow \infty} F^n$  exists, without oscillations. Note, however, that this approach is also somewhat philosophical and further analysis is possible only for special cases.

### *Proof of Result (3.88)*

The result (3.88) follows from (3.85) only if the composition operation is associative. This is indeed the case. To explore this further, note that (3.85) gives:

$$X_{n+1} = X_n \circ F \quad (i)$$

$$X_n = X_{n-1} \circ F \quad (ii)$$

Substitute (ii) in (i):

$$X_{n+1} = (X_{n-1} \circ F) \circ F \quad (iii)$$

If the composition operation is “associative” we can write (iii) as:

$$X_{n+1} = X_{n-1} \circ (F \circ F) = X_{n-1} \circ F^2 \quad (iv)$$

A simple proof for associativity of the sup-min composition is as follows. Note that sup-min composition is analogous to matrix multiplication where “min” is used in place of “product” and “sup” is used in place of “sum,” as illustrated in Chapter 2. Now, sum and product operations satisfy the following properties:

- (i) sum is associative with itself:  $(a + b) + c = a + (b + c)$
- (ii) product is associative with itself:  $(a \times b) \times c = a \times (b \times c)$
- (iii) product is distributive over sum:  $a \times (b + c) = (a \times b) + (a \times c)$

Since matrix multiplication uses sum and product operations of scalars, the above three properties are the reason for associativity of matrix multiplication.

Sup and min operations satisfy the same properties, as follows:

- (i) sup is associative with itself:  $\sup(\sup(a, b), c) = \sup(a, \sup(b, c))$
- (ii) min is associative with itself:  $\min(\min(a, b), c) = \min(a, \min(b, c))$
- (iii) min is distributive over sup:  $\min(a, \sup(b, c)) = \sup(\min(a, b), \min(a, c))$

In fact, sup is distributive over min:  $\sup(a, \min(b, c)) = \min(\sup(a, b), \sup(a, c))$ , but this property is not needed here. It follows that the composition operation is associative.

In view of this, by repeated application of (i) in (iv) we get the required result (3.88).

### Example 3.5

Consider a first order fuzzy dynamic system whose free (unforced; input = 0) response is given by

$$X_{j+1} = X_j \circ R$$

where

$X_j$  = discrete membership function (column vector) of the  $j$ -th fuzzy state of the system.

$R$  = matrix representing the fuzzy rule base relation of state transition.

Investigate the stability of this decision-making process for the following examples:

$$(i) \quad R = \begin{bmatrix} 0.5 & 0.4 \\ 0.4 & 0.5 \end{bmatrix}$$

$$(ii) \quad R = \begin{bmatrix} 0.4 & 0.5 \\ 0.5 & 0.4 \end{bmatrix}$$

$$(iii) \quad R = \begin{bmatrix} 0.5 & 0.4 \\ 0.4 & 0.4 \end{bmatrix}$$

$$(iv) \quad R = \begin{bmatrix} 0.4 & 0.4 \\ 0.5 & 0.4 \end{bmatrix}$$

$$(v) \quad R = \begin{bmatrix} 1.0 & 0.8 \\ 0.5 & 0.8 \end{bmatrix}$$

$$(vi) \quad R = \begin{bmatrix} 0.5 & 0.8 \\ 1.0 & 0.8 \end{bmatrix}$$

$$(vii) \quad R = \begin{bmatrix} 1.0 & 0.5 \\ 0.5 & 1.0 \end{bmatrix}$$

$$(viii) \quad R = \begin{bmatrix} 0.5 & 1.0 \\ 1.0 & 0.5 \end{bmatrix}$$

$$(ix) \quad R = \begin{bmatrix} 1.0 & 0.2 \\ 0.2 & 1.0 \end{bmatrix}$$

$$(x) \quad R = \begin{bmatrix} 0.2 & 1.0 \\ 1.0 & 0.2 \end{bmatrix}$$

Based on your numerical results, suggest a criterion for establishing the stability of a fuzzy decision-making system of this type.

**Solution**

For the composition operation, simply carry out matrix multiplication, replacing “product” by “min” and “sum” by “max.”

$$(i) \quad R = \begin{bmatrix} 0.5 & 0.4 \\ 0.4 & 0.5 \end{bmatrix} \quad R^2 = \begin{bmatrix} 0.5 & 0.4 \\ 0.4 & 0.5 \end{bmatrix} \circ \begin{bmatrix} 0.5 & 0.4 \\ 0.4 & 0.5 \end{bmatrix} = \begin{bmatrix} 0.5 & 0.4 \\ 0.4 & 0.5 \end{bmatrix}$$

Note the convergence of the result. Hence,

$$R^\infty = \begin{bmatrix} 0.5 & 0.4 \\ 0.4 & 0.5 \end{bmatrix} \Rightarrow \text{stable}$$

$$(ii) \quad R = \begin{bmatrix} 0.4 & 0.5 \\ 0.5 & 0.4 \end{bmatrix} \quad R^2 = \begin{bmatrix} 0.4 & 0.5 \\ 0.5 & 0.4 \end{bmatrix} \circ \begin{bmatrix} 0.4 & 0.5 \\ 0.5 & 0.4 \end{bmatrix} = \begin{bmatrix} 0.5 & 0.4 \\ 0.4 & 0.5 \end{bmatrix}$$

$$R^3 = \begin{bmatrix} 0.4 & 0.5 \\ 0.5 & 0.4 \end{bmatrix} \circ \begin{bmatrix} 0.5 & 0.4 \\ 0.4 & 0.5 \end{bmatrix} = \begin{bmatrix} 0.4 & 0.5 \\ 0.5 & 0.4 \end{bmatrix}$$

It follows that  $R^\infty$  oscillates between  $\begin{bmatrix} 0.4 & 0.5 \\ 0.5 & 0.4 \end{bmatrix}$  and  $\begin{bmatrix} 0.5 & 0.4 \\ 0.4 & 0.5 \end{bmatrix}$

This represents instability.

*Note:* In view of associativity of composition operation ( $\circ$ ), it is clear that

$$R^3 = R \circ (R \circ R) = (R \circ R) \circ R$$

This is verified since

$$R^3 = \begin{bmatrix} 0.4 & 0.5 \\ 0.5 & 0.4 \end{bmatrix} \circ \begin{bmatrix} 0.5 & 0.4 \\ 0.4 & 0.5 \end{bmatrix} = \begin{bmatrix} 0.5 & 0.4 \\ 0.4 & 0.5 \end{bmatrix} \circ \begin{bmatrix} 0.4 & 0.5 \\ 0.5 & 0.4 \end{bmatrix} = \begin{bmatrix} 0.4 & 0.5 \\ 0.5 & 0.4 \end{bmatrix}$$

$$(iii) \quad R = \begin{bmatrix} 0.5 & 0.4 \\ 0.4 & 0.4 \end{bmatrix} \quad R^2 = \begin{bmatrix} 0.5 & 0.4 \\ 0.4 & 0.4 \end{bmatrix} \circ \begin{bmatrix} 0.5 & 0.4 \\ 0.4 & 0.4 \end{bmatrix} = \begin{bmatrix} 0.5 & 0.4 \\ 0.4 & 0.4 \end{bmatrix}$$

The result converges. Hence

$$R^\infty = \begin{bmatrix} 0.5 & 0.4 \\ 0.4 & 0.4 \end{bmatrix} \Rightarrow \text{stable}$$

$$(iv) \quad R = \begin{bmatrix} 0.4 & 0.4 \\ 0.5 & 0.4 \end{bmatrix} \quad R^2 = \begin{bmatrix} 0.4 & 0.4 \\ 0.5 & 0.4 \end{bmatrix} \circ \begin{bmatrix} 0.4 & 0.4 \\ 0.5 & 0.4 \end{bmatrix} = \begin{bmatrix} 0.4 & 0.4 \\ 0.4 & 0.4 \end{bmatrix}$$

$$R^3 = \begin{bmatrix} 0.4 & 0.4 \\ 0.5 & 0.4 \end{bmatrix} \circ \begin{bmatrix} 0.4 & 0.4 \\ 0.4 & 0.4 \end{bmatrix} = \begin{bmatrix} 0.4 & 0.4 \\ 0.4 & 0.4 \end{bmatrix}$$

The result converges. Hence

$$R^\infty = \begin{bmatrix} 0.4 & 0.4 \\ 0.4 & 0.4 \end{bmatrix} \Rightarrow \text{stable}$$

$$(v) \quad R = \begin{bmatrix} 1.0 & 0.8 \\ 0.5 & 0.8 \end{bmatrix} \quad R^2 = \begin{bmatrix} 1.0 & 0.8 \\ 0.5 & 0.8 \end{bmatrix} \circ \begin{bmatrix} 1.0 & 0.8 \\ 0.5 & 0.8 \end{bmatrix} = \begin{bmatrix} 1.0 & 0.8 \\ 0.5 & 0.8 \end{bmatrix}$$

The result converges. Hence

$$R^\infty = \begin{bmatrix} 1.0 & 0.8 \\ 0.5 & 0.8 \end{bmatrix} \Rightarrow \text{stable}$$

$$(vi) \quad R = \begin{bmatrix} 0.5 & 0.8 \\ 1.0 & 0.8 \end{bmatrix} \quad R^2 = \begin{bmatrix} 0.5 & 0.8 \\ 1.0 & 0.8 \end{bmatrix} \circ \begin{bmatrix} 0.5 & 0.8 \\ 1.0 & 0.8 \end{bmatrix} = \begin{bmatrix} 0.8 & 0.8 \\ 0.8 & 0.8 \end{bmatrix}$$

$$R^3 = \begin{bmatrix} 0.5 & 0.8 \\ 1.0 & 0.8 \end{bmatrix} \circ \begin{bmatrix} 0.8 & 0.8 \\ 0.8 & 0.8 \end{bmatrix} = \begin{bmatrix} 0.8 & 0.8 \\ 0.8 & 0.8 \end{bmatrix}$$

The result converges. Hence

$$R^\infty = \begin{bmatrix} 0.8 & 0.8 \\ 0.8 & 0.8 \end{bmatrix} \Rightarrow \text{stable}$$

$$(vii) \quad R = \begin{bmatrix} 1.0 & 0.5 \\ 0.5 & 1.0 \end{bmatrix} \quad R^2 = \begin{bmatrix} 1.0 & 0.5 \\ 0.5 & 1.0 \end{bmatrix} \circ \begin{bmatrix} 1.0 & 0.5 \\ 0.5 & 1.0 \end{bmatrix} = \begin{bmatrix} 1.0 & 0.5 \\ 0.5 & 1.0 \end{bmatrix}$$

The result converges. Hence

$$R^\infty = \begin{bmatrix} 1.0 & 0.5 \\ 0.5 & 1.0 \end{bmatrix} \Rightarrow \text{stable}$$

$$(viii) \quad R = \begin{bmatrix} 0.5 & 1.0 \\ 1.0 & 0.5 \end{bmatrix} \quad R^2 = \begin{bmatrix} 0.5 & 1.0 \\ 1.0 & 0.5 \end{bmatrix} \circ \begin{bmatrix} 0.5 & 1.0 \\ 1.0 & 0.5 \end{bmatrix} = \begin{bmatrix} 1.0 & 0.5 \\ 0.5 & 1.0 \end{bmatrix}$$

$$R^3 = \begin{bmatrix} 0.5 & 1.0 \\ 1.0 & 0.5 \end{bmatrix} \circ \begin{bmatrix} 1.0 & 0.5 \\ 0.5 & 1.0 \end{bmatrix} = \begin{bmatrix} 0.5 & 1.0 \\ 1.0 & 0.5 \end{bmatrix}$$

It follows that  $R^\infty$  oscillates between  $\begin{bmatrix} 0.5 & 1.0 \\ 1.0 & 0.5 \end{bmatrix}$  and  $\begin{bmatrix} 1.0 & 0.5 \\ 0.5 & 1.0 \end{bmatrix}$

This is an unstable situation.

$$(ix) \quad R = \begin{bmatrix} 1.0 & 0.2 \\ 0.2 & 1.0 \end{bmatrix} \quad R^2 = \begin{bmatrix} 1.0 & 0.2 \\ 0.2 & 1.0 \end{bmatrix} \circ \begin{bmatrix} 1.0 & 0.2 \\ 0.2 & 1.0 \end{bmatrix} = \begin{bmatrix} 1.0 & 0.2 \\ 0.2 & 1.0 \end{bmatrix}$$

The result converges. Hence

$$R^\infty = \begin{bmatrix} 1.0 & 0.2 \\ 0.2 & 1.0 \end{bmatrix} \Rightarrow \text{stable}$$

$$(x) \quad R = \begin{bmatrix} 0.2 & 1.0 \\ 1.0 & 0.2 \end{bmatrix} \quad R^2 = \begin{bmatrix} 0.2 & 1.0 \\ 1.0 & 0.2 \end{bmatrix} \circ \begin{bmatrix} 0.2 & 1.0 \\ 1.0 & 0.2 \end{bmatrix} = \begin{bmatrix} 1.0 & 0.2 \\ 0.2 & 1.0 \end{bmatrix}$$

$$R^3 = \begin{bmatrix} 0.2 & 1.0 \\ 1.0 & 0.2 \end{bmatrix} \circ \begin{bmatrix} 1.0 & 0.2 \\ 0.2 & 1.0 \end{bmatrix} = \begin{bmatrix} 0.2 & 1.0 \\ 1.0 & 0.2 \end{bmatrix}$$

It is seen that  $R^\infty$  oscillates between  $\begin{bmatrix} 0.2 & 1.0 \\ 1.0 & 0.2 \end{bmatrix}$  and  $\begin{bmatrix} 1.0 & 0.2 \\ 0.2 & 1.0 \end{bmatrix}$

This is an unstable situation.

From the results it appears that, when  $R$  is diagonally strong, the system is stable; when  $R$  is off-diagonally strong, the system is oscillatory (unstable).

### 3.6.3 Eigen-fuzzy sets

Analogous to how eigenvalues and eigenvectors can predict the natural response and stability of a crisp, linear system, it is expected that the eigen-fuzzy sets will provide useful information concerning stability of a fuzzy decision-making system.

For a given fuzzy relation (rule base)  $R(x, y)$ , its eigen-fuzzy sets are defined as the family of fuzzy sets  $E$  such that

$$E = \{E | E \circ R = E\}$$

where “ $\circ$ ” denotes the composition operation. Note here the analogy to eigenvectors of a matrix. Also, note that

$$R: X \times X \rightarrow [0, 1]$$

#### 3.6.3.1 Iterative method

An iterative method to obtain the greatest eigen-fuzzy set of  $R$  is given below.

Initialize:  $\mu_{E_1}(x) = \sup_{y \in Y} \mu_R(x, y)$

Iterate:

$$E_2 = E_1 \circ R$$

$$E_3 = E_2 \circ R$$

$$E_4 = E_3 \circ R$$

...

$$E_{k+1} = E_k \circ R$$

until the result is sufficiently converged (i.e.,  $\mu_{E_{k+1}} \approx \mu_{E_k}$ ).  
Similarly,

Initialize:  $\mu_{E_1}(y) = \sup_{x \in X} \mu_R(x, y)$

Iterate:

$$E_2 = E_1 \circ R$$

$$E_3 = E_2 \circ R$$

$$E_4 = E_3 \circ R$$

...

$$E_{k+1} = E_k \circ R$$

until convergence.

### Example 3.6

The application of the iterative approach is illustrated now using an example.  
Consider the fuzzy rule base membership function (discrete and binary):

$$R(x_i, y_j) = \begin{bmatrix} 1.0 & 0.2 \\ 0.4 & 0.6 \end{bmatrix}$$

Then

$$\mu_{E_1}(x_i) = \max_{y_j} \mu_R(x_i, y_j) = \begin{bmatrix} 1.0 \\ 0.6 \end{bmatrix}$$

Now using *sup-min* composition,

$$[1.0 \quad 0.6] \circ \begin{bmatrix} 1.0 & 0.2 \\ 0.4 & 0.6 \end{bmatrix} = [1.0 \quad 0.6]$$

where “◦” denotes the *sup-min* operation for vectors and matrices.

Hence  $\begin{bmatrix} 1.0 \\ 0.6 \end{bmatrix}$  is an eigen-fuzzy set of  $R$ .

Alternatively,

$$\mu_{E_1}(y_j) = \max_{x_i} \mu_R(x_i, y_j) = [1.0 \quad 0.6]$$

Then using *sup-min* composition,

$$[1.0 \quad 0.6] \circ R^T(x_i, y_j) = [1.0 \quad 0.6] \circ \begin{bmatrix} 1.0 & 0.4 \\ 0.2 & 0.6 \end{bmatrix} = [1.0 \quad 0.6]$$

Again,  $[1.0 \quad 0.6]$  is an eigen-fuzzy set of  $R$ .

### 3.7 Summary

Fuzzy logic control or simply “fuzzy control” belongs to the class of “intelligent control,” “knowledge-based control,” or “expert control.” Fuzzy control uses knowledge-based decision-making employing techniques of fuzzy logic, as presented in Chapter 2, in determining the control actions. In this chapter, the basics of analysis and application of fuzzy logic control were presented. The method of generating an aggregate membership function for a fuzzy rule base was studied. The process of making control decisions by applying the *compositional rule of inference* was explained. Typically the monitored process variables are crisp and they have to be *fuzzified* for the purpose of decision-making. Also, the control decisions generated in this manner are fuzzy quantities, and they have to be *defuzzified* prior to using them in a physical action of control. These methods were given in this chapter. Analytical characteristics and stability considerations of fuzzy control were explored. Hierarchical control corresponds to an architecture that is commonly used in distributed control systems, and is particularly relevant when various control functions in the system can be conveniently ranked into different levels. In this chapter, some attempt was made to differentiate between high-level control and low-level direct control in a hierarchical control system. It was hinted that fuzzy logic would be more suited for the former. Examples were given to illustrate the various steps of fuzzy logic control.

### Problems

- Suppose that the state of “fast speed” of a machine is denoted by the fuzzy set  $F$  with membership function  $\mu_F(v)$ . Then the state of “very fast speed,” where the *linguistic hedge* “very” has been incorporated, may be represented by  $\mu_F(v - v_0)$  with  $v_0 > 0$ . Also, the state “presumably fast speed,” where the linguistic hedge “presumably” has been incorporated, may be represented by  $\mu_F^2(v)$ .

Discuss the appropriateness of the use of these membership functions to represent the respective linguistic hedges.

In particular, if

$$F = \left\{ \frac{0.1}{10}, \frac{0.3}{20}, \frac{0.6}{30}, \frac{0.8}{40}, \frac{1.0}{50}, \frac{0.7}{60}, \frac{0.5}{70}, \frac{0.3}{80}, \frac{0.1}{90} \right\}$$

in the discrete universe  $V = \{0, 10, 20, \dots, 190, 200\}$  rev/s and  $v_0 = 50$  rev/s, determine the membership functions of “very fast speed” and “presumably fast speed.”

Suppose that power  $p$  is given by the relation (crisp)

$$p = v^2$$

For the given fast speed (fuzzy)  $F$ , determine the corresponding membership function for power. Compare/contrast this with “presumably fast speed.”

2. Two fuzzy sets  $A$  and  $B$  are represented by the membership functions shown in Figure P3.2. Sketch the membership functions of the following fuzzy sets/relations:

$$(a) A \cup B; \quad (b) A \cap B; \quad (c) \text{NOT } A; \quad (d) A \rightarrow B.$$

In (a) and (b) assume that  $A$  and  $B$  are defined in the same universe  $X = [0, 8]$  and in (d) assume that  $B$  is defined in another universe  $Y$ . Both universes are subsets of the real line  $\mathbb{R}$ .

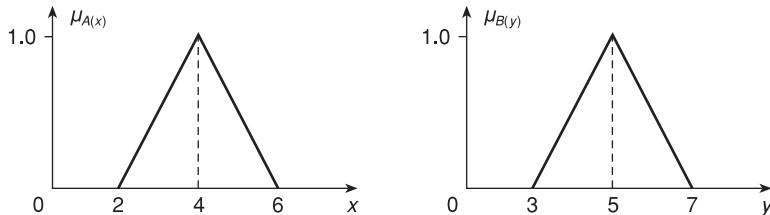


Figure P3.2: Membership functions of two fuzzy sets

In case (d) suppose that a crisp reading of  $A'$  is observed as  $x = 5$ . What is the membership function of the corresponding inference  $B'$ ? If this inference is defuzzified using the centroid method, what is the crisp action?

3. Consider the experimental setup of an inverted pendulum shown in Figure P3.3.

Suppose that direct fuzzy logic control is used to keep the inverted pendulum upright. The process measurements are the angular position, about the vertical ( $ANG$ ) and the angular velocity ( $VEL$ ) of the pendulum. The control action ( $CNT$ ) is the current of the motor driving the positioning trolley. The variable  $ANG$  takes two fuzzy states: positive large ( $PL$ ) and negative large ( $NL$ ). Their memberships are defined in the support set  $[-30^\circ, 30^\circ]$  and are trapezoidal. Specifically,

$$\begin{aligned} \mu_{PL} &= 0 && \text{for } ANG = [-30^\circ, -10^\circ] \\ &= \text{linear } [0, 1.0] && \text{for } ANG = [-10^\circ, 20^\circ] \\ &= 1.0 && \text{for } ANG = [20^\circ, 30^\circ] \end{aligned}$$

$$\begin{aligned} \mu_{NL} &= 1.0 && \text{for } ANG = [-30^\circ, -20^\circ] \\ &= \text{linear } [1.0, 0] && \text{for } ANG = [-20^\circ, 10^\circ] \\ &= 0 && \text{for } ANG = [10^\circ, 30^\circ] \end{aligned}$$

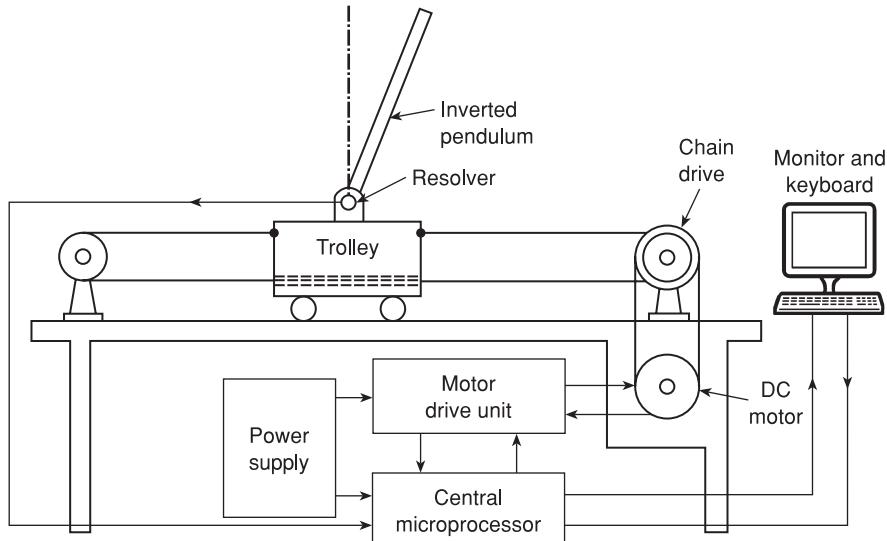


Figure P3.3: A computer-controlled inverted pendulum

The variable  $VEL$  takes two fuzzy states  $PL$  and  $NL$ , which are quite similarly defined in the support set  $[-60^\circ/\text{s}, 60^\circ/\text{s}]$ . The control inference  $CNT$  can take three fuzzy states: positive large ( $PL$ ), no change ( $NC$ ), and negative large ( $NL$ ). Their membership functions are defined in the support set  $[-3\text{A}, 3\text{A}]$  and are either trapezoidal or triangular. Specifically,

$$\begin{aligned}
 \mu_{PL} &= 0 && \text{for } CNT = [-3\text{A}, 0] \\
 &= \text{linear } [0, 1.0] && \text{for } CNT = [0.2\text{A}] \\
 &= 1.0 && \text{for } CNT = [2\text{A}, 3\text{A}] \\
 \mu_{NC} &= 0 && \text{for } CNT = [-3\text{A}, -2\text{A}] \\
 &= \text{linear } [0, 1.0] && \text{for } CNT = [-2\text{A}, 0] \\
 &= \text{linear } [1.0, 0] && \text{for } CNT = [0, 2\text{A}] \\
 &= 0 && \text{for } CNT = [2\text{A}, 3\text{A}] \\
 \mu_{NL} &= 1.0 && \text{for } CNT = [-3\text{A}, -2\text{A}] \\
 &= \text{linear } [1.0, 0] && \text{for } CNT = [-2\text{A}, 0] \\
 &= 0 && \text{for } CNT = [0, 3\text{A}]
 \end{aligned}$$

The following four fuzzy rules are used in control:

```

if ANG is PL and VEL is PL then CNT is NL
else if ANG is PL and VEL is NL then CNT is NC
else if ANG is NL and VEL is PL then CNT is NC
else if NAG is NFL and EL is NFL then CT is LP
end if.
  
```

- Sketch the four rules in a membership diagram for the purpose of making control inferences using individual rule-based inference.
- If the process measurements of  $ANG = 5^\circ$  and  $VEL = 15^\circ/\text{s}$  are made, indicate on your sketch the corresponding control inference.

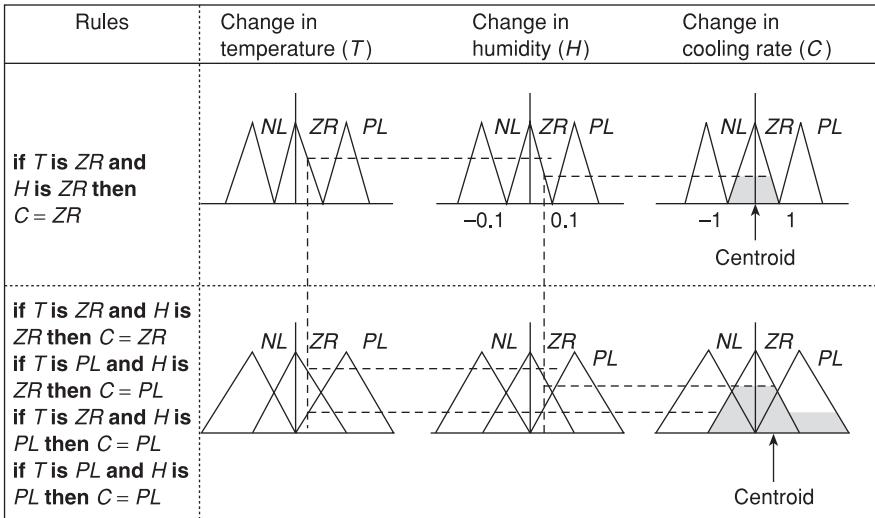


Figure P3.4: The effect of membership overlap on the validity of control inference

4. Consider again the comfort control system discussed in this chapter. Suppose that some of the rules in the control knowledge base are

```

If      T   is   ZR   and   H   is   ZR   then   C   is   NC
else if   T   is   PL   and   H   is   ZR   then   C   is   PL
else if   T   is   ZR   and   H   is   PL   then   C   is   PL
else if   T   is   PL   and   H   is   PL   then   C   is   PL
else if ...
  
```

Note that  $T$  = Change in temperature from reference

$H$  = Change in humidity from reference

$C$  = Change in cooling rate from reference.

Two cases can be considered:

- The membership functions do not overlap.
- The membership functions have some overlap.

These two cases are illustrated in Figure P3.4. Discuss why case (b) provides more realistic control inferences than case (a). You may consider the inference  $C$  produced in the two cases for a specific reading of  $T$  and  $H$ .

5. Many recent papers on fuzzy logic control use simple, linear models to represent the plants, thereby contravening the purpose of using knowledge-based control in the first place. In one such paper, the context variables of the control rule base are taken as the response error, the summation of the error, and the change in error, somewhat analogously to the classic proportional-integral-derivative (PID) control. The work goes on to assume that the behavior of the control system would be similar to what one normally observes under PID control, for example, in terms of stability, speed of response, and steady-state accuracy. Discuss the validity of such conclusions.

6. Considering an intelligent controller as a real-time expert system, and reading pertinent literature, briefly explain the following terms:
- Knowledge engineering interface
  - Knowledge validation
  - Explanation of reasoning
  - Truth maintenance
  - Dynamic knowledge base
  - Real-time performance
  - Uncertainty management.
7. Does “fuzzification” of a crisp item of data generally lead to loss of information? Carefully justify your answer. You may use a statistical analogy.
8. Consider the freeway incident-detection problem as outlined in [Hall, L., Hall, F.L., and Shi, Y (1991). A Catastrophe Theory Approach to Freeway Incident Detection, *Proc. 2nd Int. Conf. on Applications of Advanced Technologies in Transportation Engineering*, New York, NY, pp. 373–7]. Use the following definitions:

Flow rate  $f$  = number of vehicles passing a specified point of a lane, per unit time

$$\text{Occupancy } d = \frac{\text{Number of vehicles in a given segment of a lane} \times 100\%}{\text{Maximum number of vehicles the segment can occupy}}$$

Note that discrepancies can result if the vehicle parameters (geometric and kinematic) and road parameters (road conditions, grade, etc.) are not uniform or fixed.

It is required to establish a lower-bound curve for uncongested flow on the “Flow-Rate versus Occupancy” plane such that points above the curve represent uncongested flow and the points below represent congested flow. Figure P3.8 shows a schematic representation of a set of experimental data (occupancy, flow) pairs for a lane segment of freeway. Here  $u$  denotes a data point of uncongested flow and  $c$  denotes one of congested flow. It is noted that some overlap is present, and hence it is difficult to establish a single lower-bound curve.

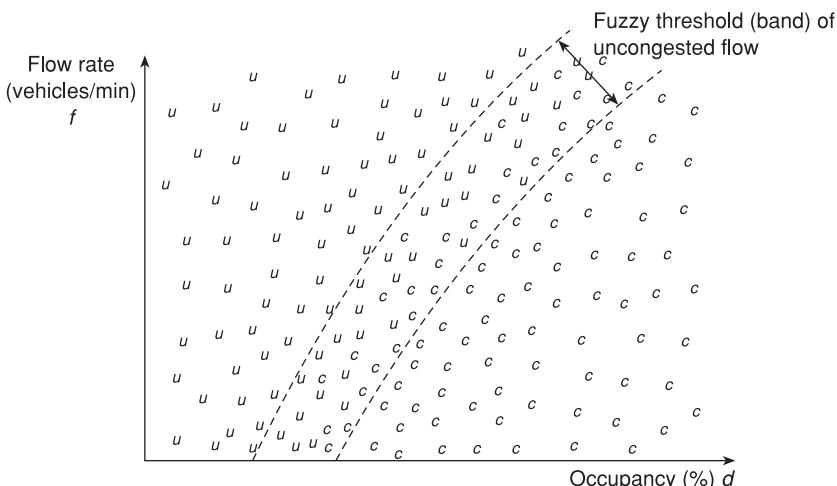


Figure P3.8: Occupancy-flow information for a freeway lane segment

- (a) Give reasons for such overlaps.

Suppose that a fuzzy representation of the lower-bound curve is considered within the overlap band. Specifically, curves of the form

$$f = bd^{1/n} + a \quad (i)$$

are considered with fixed parameters  $b$  and  $n$  (say,  $b = 6.7$  and  $n = 0.5$ ) and the parameter  $a$  being fixed for each curve. The units of  $f$  are vehicles/min, and  $d$  is expressed as a percentage. It is required to estimate a membership function  $\mu_U(a)$  such that the membership grade for a particular value of  $a$  would represent the possibility that the corresponding curve given by equation (i) falls into the uncongested region. In particular, if  $\mu_U(a) = 0$  the curve would be completely in the congested region, and if  $\mu_U(a) = 1$  it would be completely in the uncongested region.

- (b) Suggest a way to estimate this membership function, using the experimental data that are schematically shown in the figure, say for values of  $a$  in the range  $[-70, 20]$ .  
(c) Discuss whether your estimate can be more appropriately considered as either a *belief function* or a *plausibility function* rather than a *membership function*.

9. A fuzzy logic rule, for example, as given by

$$R_i: (A_i, B_i) \rightarrow C_i$$

is called a “fuzzy association.” A *fuzzy associative memory* (FAM) is formed by partitioning the universe of discourse of each condition variable (i.e.,  $A_i$  and  $B_i$  in the present example) according to the level of *fuzzy resolution* (number of fuzzy states) chosen for these antecedents, thereby generating a grid of FAM elements. The entry at each grid element in the FAM corresponds to the fuzzy action ( $C_i$  in the example). A fuzzy associate memory may be, then, interpreted as a geometric or tabular representation of a fuzzy logic rule base.

A fuzzy associate memory of a fuzzy logic controller is shown in the FAM diagram in Table P3.9. Express these rules in a linguistic form. What do the empty entries in the table (matrix) represent?

**Table P3.9:** FAM diagram of a fuzzy control rule base

		Change in error $\Delta e$				
		NL	NS	ZO	PS	PL
Error $e$	PL		NS	NL	NL	NL
	PS		ZO	NS	NL	NL
	ZO	PL	PS	ZO	NS	NL
	NS	PL	PL	PS	ZO	
	NL	PL	PL	PL	PS	

Note: Here, N denotes “negative,” P denotes “positive,” S denotes “small,” L denotes “large,” and ZO denotes “zero.”

10. In an attempt to compare the “centroid method” and the “mean of max method” of defuzzification, suppose that the control inference  $C'$  has a triangular membership function with  $\mu_{C'}(c)$  with a support set  $[-c_0, c_0]$  and a peak value of 1.0 at  $c = c_0/2$ . A thresholded defuzzification is used with a threshold value of  $\alpha$ . Show that the defuzzified control action  $\hat{c}$  is given by

$$\hat{c} = \frac{c_0}{2} \text{ for all } \alpha, \text{ in the mean of max method}$$

$$= \frac{c_0}{6} (1 + 2\alpha), \text{ in the centroid method}$$

11. Consider a fuzzy rule base in which each rule is of the form

If  $X_i$  And  $Y_i$  Then  $Z_i$

in which  $X_i$  And  $Y_i$  are fuzzy conditions and  $Z_i$  is a fuzzy action. The complete rule base is given by

$$R: OR_i[X_i \text{ AND } Y_i \rightarrow Z_i]$$

In terms of the membership functions of  $X_i$ ,  $Y_i$  and  $Z_i$ , the membership function  $\mu_{R_i}$  of each rule is normally obtained by

$$\mu_{R_i}(x, y, z) = \min[\mu_{X_i}(x), \mu_{Y_i}(y), \mu_{Z_i}(z)] \quad (\text{i})$$

and the membership function  $\mu_R$  of the entire rule base is given by

$$\mu_R(x, y, z) = \max_i \mu_{R_i}(x, y, z) \quad (\text{ii})$$

Instead of using equation (ii) to determine  $\mu_R$ , suppose that the membership of each rule is modified by introducing a rule parameter  $\alpha_i$  such that

$$\mu_R(x, y, z) = \max_i [\mu_{R_i} + \alpha_i \mu_{R_i}(1 - \mu_{R_i})] \quad (\text{ii})*$$

in which  $\mu_{R_i}$  is obtained using equation (i), as before. Also,  $-1 \leq \alpha_i \leq 1$ .

Explain the physical significance of the rule parameter  $\alpha_i$ . In particular, for the cases of  $\alpha_i > 0$  and  $\alpha_i < 0$ , respectively, what does  $\alpha_i$  offer to the rule-based decision-making process?

12. (i) Which control method would you recommend for each of the following applications:
- Servo control of a single-axis positioning table with a permanent-magnet DC motor (linear).
  - Active control of a vehicle suspension system (linear, multivariable).
  - Control of a rotary cement kiln (nonlinear, complex, difficult to model).
- (ii) A metallurgical process consists of heat treatment of a bulk of material for a specified duration of time at a suitable temperature. The heater is controlled by its fuel supply rate. A schematic diagram of the system is shown in Figure P3.12(a).

The following fuzzy quantities are defined, with the corresponding states:

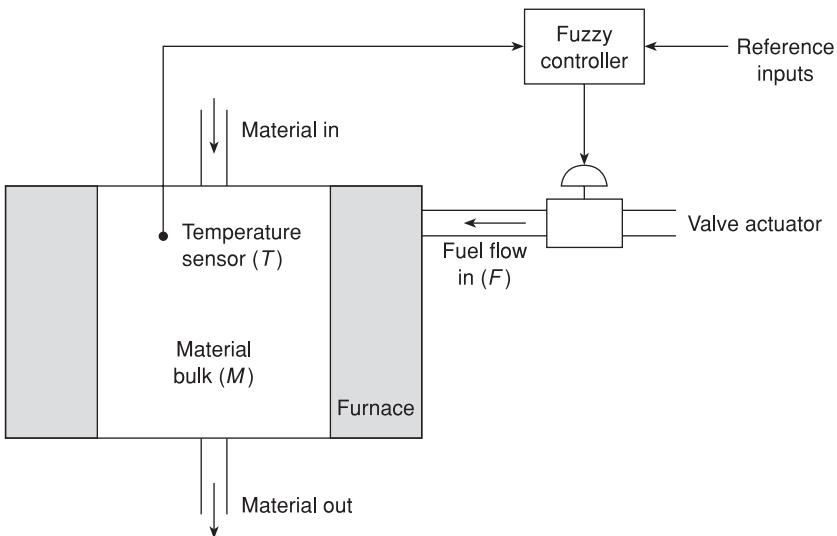
$T$ : Temperature of the material ( $LW$  = low;  $HG$  = high)

$M$ : Mass of the material ( $SM$  = small;  $LG$  = large)

$P$ : Process termination time ( $FR$  = far;  $NR$  = near)

$F$ : Fuel supply rate ( $RD$  = reduce;  $MN$  = maintain;  $IN$  = increase)

The membership functions of these quantities are given in Figure P3.12(b). A simple rule base that is used in a fuzzy controller for the fuel supply unit is given below:



**Figure P3.12 (a):** A metallurgical heat treatment process

If  $T$  is *LW* and  $P$  is *FR* then  $F$  is *IN*  
 or if  $T$  is *HG* then  $F$  is *RD*  
 or if  $M$  is *SM* and  $P$  is *NR* then  $F$  is *MN*  
 or if  $M$  is *LG* and  $P$  is *FR* then  $F$  is *IN*  
 or if  $P$  is *NR* then  $F$  is *RD*  
 End if.

At a given instant, the following set of process data is available:

Temperature =  $300^{\circ}\text{C}$   
 Material mass = 800 kg  
 Process operation time = 1.3 hr

Determine the corresponding inference membership function for the fuel supply, and a crisp value for the control action. Comment on the suitability of this inference.

- 13.** Clearly indicate why, for *max-product composition* (max-dot composition) denoted by “ $\circ$ ” it is true that

$$[A(x_i) \circ P(x_i, y_j)] \circ Q(y_j, z_k) = A(x_i) \circ [P(x_i, y_j) \circ Q(y_j, z_k)]$$

Discuss the implication and a practical use of this result in fuzzy decision-making; for example, sequential control.

- 14.** This problem deals with rule interaction in fuzzy logic control. First consider just one rule,

$$R_1: A_1 \rightarrow C_1$$

expressed in the discrete form:

$$A_1 = 0.7/a_1 + 0.6/a_2 + 0.1/a_3 \text{ with cardinality 3}$$

$$C_1 = 0.5/c_1 + 0.4/c_2 \text{ with cardinality 2}$$

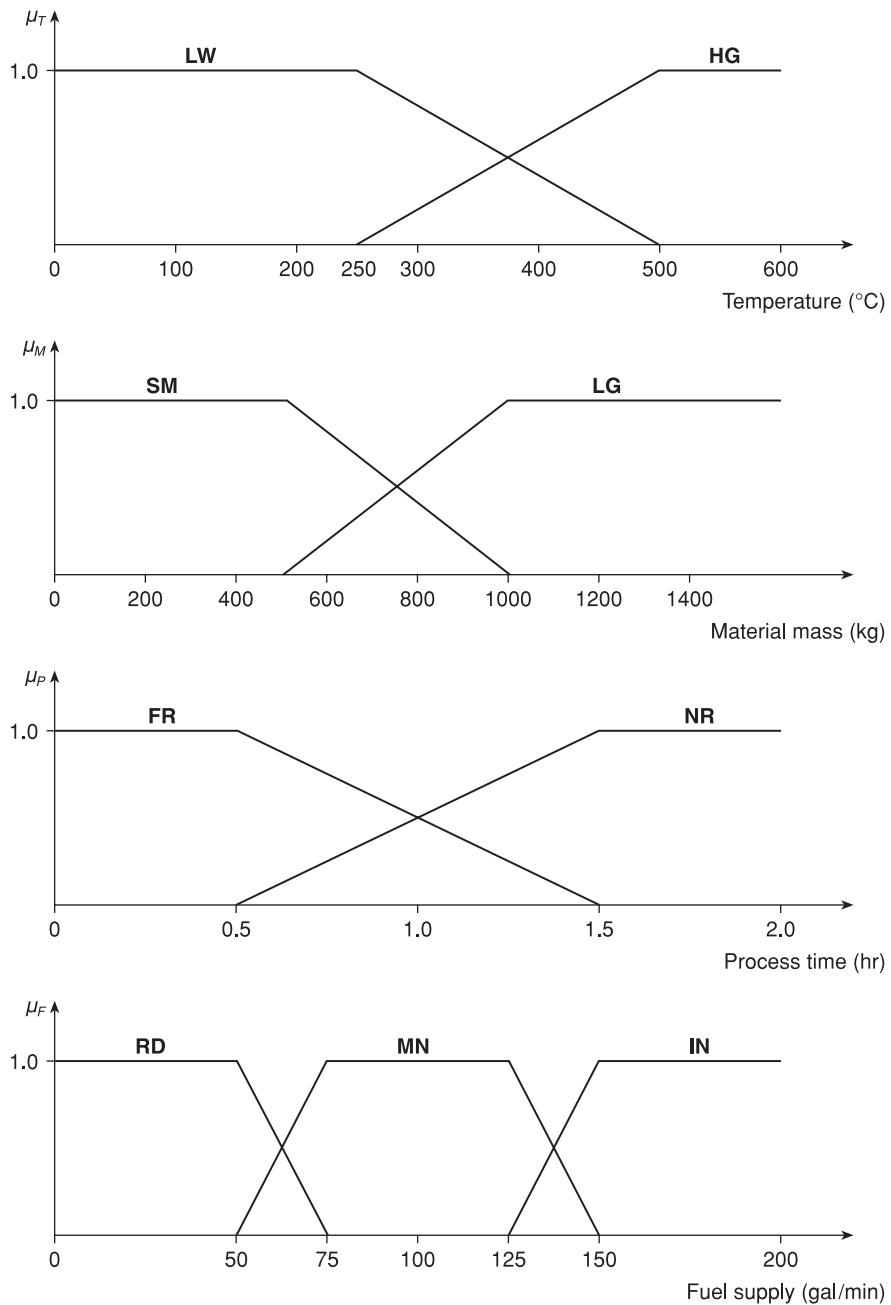


Figure P3.12 (b): Membership functions

Show that this rule base may be expressed as

$$\mu_{R1}(a_i, c_j) = \begin{bmatrix} 0.5 & 0.4 \\ 0.5 & 0.4 \\ 0.1 & 0.1 \end{bmatrix}$$

Now suppose that a fuzzy observation

$$A'_1 = 0.7/a_1 + 0.6/a_2 + 0.1/a_3$$

is given. Show by applying the max-min composition that the corresponding control inference is given by

$$C' = 0.5/c_1 + 0.4/c_2$$

Next suppose that a second rule  $R_2$  is available,

$$R_2: A_2 \rightarrow C_2$$

with

$$A_2 = 0.6/a_1 + 0.7/a_2 + 0.2/a_3$$

$$C_2 = 0.4/c_1 + 0.5/c_2$$

Show that the membership function of the combined rule base  $R = R_1 \vee R_2$  is given by

$$\mu_R(a_i, c_j) = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \\ 0.2 & 0.2 \end{bmatrix}$$

Now, for a fuzzy observation

$$A'_1 = 0.7/a_1 + 0.6/a_2 + 0.1/a_3$$

which is the same as before, show that the corresponding control inference is

$$C' = 0.5/c_1 + 0.5/c_2$$

which is different from the previous inference.

15. Consider the knowledge base given in the bullet list below for tuning a proportional-integral-derivative (PID) controller.
- If the error response is not oscillatory *then* do not change the proportional and derivative control parameters;  
or if the error response is moderately oscillatory *then* make a small decrease in the proportional gain and a small increase in the derivative time constant;  
or if the error response is highly oscillatory *then* make a large decrease in the proportional gain and a large increase in the derivative time constant.
  - If the error response is fast enough *then* do not change the proportional and derivative control parameters;  
or if the error response is not fast enough *then* make a small increase in the proportional gain and a small increase in the derivative time constant.
  - If the error response does not steadily diverge *then* do not change the proportional and integral and derivative control parameters; or if the error response steadily diverges *then* slightly decrease the proportional gain and slightly decrease the integral rate and make a large increase in the derivative time constant.

- If the error response does not have an offset *then* do not change the proportional and integral control parameters;  
*or if* the error response has an offset *then* slightly increase the proportional gain and make a large increase in the integral rate.

It may be condensed to the form given below, with the notation that follows. Suppose that the membership functions (discrete) of the condition variables in the rule base are as given in Table P3.15(a), and the membership functions of the action variables are as given in Table P3.15(b). Develop a decision table for on line fuzzy tuning of the PID controller.

```

If OSC = OKY   then DP = NC
                  and DD = NC
or  If OSC = MOD then DP = NL
                  and DD = PL
or  If OSC = HIG  then DP = NH
                  and DD = PH
    If RSP = OKY  then DP = NC
                  and DD = PL
    If DIV = OKY   then DP = NC
                  and DI = NC
                  and DD = NC
or  If DIV = NOK  then DP = NL
                  and DI = NL
                  and DD = PH
    If OFF = OKY   then DP = NC
                  and DI = NC
or  If OFF = NOK  then DP = PL
                  and DI = PH

```

The notation used for the fuzzy quantities:

OSC = Oscillations in the error response  
 RSP = Speed of response (decay) of the error  
 DIV = Divergence of the error response  
 OFF = Offset in the error response  
 OKY = Satisfactory  
 MOD = Moderately unsatisfactory  
 HIG = Highly unsatisfactory  
 NOK = Unsatisfactory  
 DP = Change (relative) of the proportional gain  
 DI = Change (relative) of the integral rate  
 DD = Change (relative) of the derivative time constant  
 NH = Negative high (magnitude)  
 NL = Negative low (magnitude)  
 NC = No change  
 PL = Positive low  
 PH = Positive high

**Table P3.15 (a): Membership functions of the condition variables**

1. OSC:

	0	1	2
OKY	1.0	0.2	0.1
MOD	0.2	1.0	0.2
HIG	0.1	0.2	1.0

2. RSP:

	0	1
OKY	1.0	0.2
NOK	0.2	1.0

3. DIV:

	0	1
OKY	1.0	0.1
NOK	0.1	1.0

4. OFF:

	0	1
OKY	1.0	0.2
NOK	0.2	1.0

**Table P3.15 (b): Membership functions of the action variables**

1. DP:

	-2	-1	0	1
NH	1.0	0.2	0.1	0.0
NL	0.2	1.0	0.2	0.1
NC	0.1	0.2	1.0	0.2
PL	0.0	0.1	0.2	1.0

2. DI:

	-1	0	2
NL	1.0	0.2	0.0
NC	0.2	1.0	0.1
PH	0.0	0.1	0.0

3. DD:

	-1	0	1	2
NL	1.0	0.2	0.1	0.0
NC	0.2	1.0	0.2	0.1
PL	0.1	0.2	1.0	0.2
PH	0.0	0.1	0.2	1.0

16. It is argued that the fuzzy logic approach is not appropriate for low-level servo loop control. People tend to have very high expectations of fuzzy logic control as a

universal control approach. This may also be the reason why they hasten to criticize the approach. Discuss why fuzzy logic control may not be appropriate when the control objective is

$$e = 0 \quad \text{and} \quad \dot{e} = 0$$

where  $e$  is the response error.

- 17.** List several benefits of intelligent process control. If the process task can be decomposed into a succession of basic operations and the control requirement can be represented in terms of a set of subcontrollers, depending on the needed level of intelligence, suggest a suitable system structure for control and communication purposes.

- 18.** Stability of a system may be monitored on line in several ways.

For example, one may:

- (a) identify a linear state-space (time-domain) model using input–output data and compute its eigenvalues.
- (b) identify a transfer-function (frequency domain) model using input–output data and establish relative stability (say, using gain and phase margins).
- (c) observe the trends of the process output (response) and qualitatively establish stability.
- (d) inject a known disturbance to an input signal and observe the process response and establish stability by evaluating the change in response.

Discuss which of these approaches are appropriate for use in a fuzzy logic-based self-tuning system.

- 19.** Consider a problem of decision-making through fuzzy logic. As the precision of the available data (context) increases what is its implication on:

- (a) Quality of the inference?
- (b) Cost of the decision-making process?

- 20.** Consider the process error  $e$ , its rate  $\dot{e}$ , and so on, which may be monitored for control and self-tuning purposes. In self-tuning one may first use a knowledge-based preprocessor to evaluate the error variables, say with respect to a *reference model*, and arrive at response features such as stability; say,

$$(e, \dot{e}) \rightarrow A$$

Then a tuning rule base may be used, with preprocessed features as condition variables and tuning operations as actions; say,

$$A \rightarrow T$$

Alternatively, in self-tuning control, one may use an uncoupled rule base which relates individual error variables to separate tuning actions; say,

$$e \rightarrow T_1$$

$$\dot{e} \rightarrow T_2$$

But in low-level direct (in-loop) control that uses a rule base, such approaches as given above are normally unacceptable, and instead combined rules are used; for example,

$$(e, \dot{e}) \rightarrow c$$

Discuss the rationale for this distinction.

21. Consider the single-condition, single-action rule base given by

$$\bigcup_i (E_i \rightarrow C_i)$$

where  $i$  denotes the  $i$ -th fuzzy state (e.g., small, medium, large) of the condition variable  $E$  (e.g., response error) and  $C_i$  denotes the corresponding fuzzy state of the action variable  $C$  (e.g., control or tuning action). For a given crisp observation (measurement)  $\hat{e}$  of the process, the corresponding corrective action as inferred by a conventional fuzzy logic system may be outlined as follows:

- (a) Determine the membership grade  $\mu_i$  of the element  $\hat{e}$  in the fuzzy set  $E_i$  by reading off the value  $\mu_i = \mu_{E_i}(\hat{e})$  for all  $i$ .
- (b) Determine the fuzzy action  $\hat{C}_i$  corresponding to  $\hat{e}$ , as inferred by the  $i$ -th rule of the rule base. This is obtained by clipping  $\mu_{C_i}(c)$  at the height  $\mu_i$ ; thus,

$$\mu_{\hat{C}_i}(c) = \min(\mu_i, \mu_{C_i}(c))$$

*Note:* Here the *min* operation is used for “implication.” Alternatively, the “*dot*” (or *product*) operation may be used as

$$\mu_{\hat{C}_i}(c) = \mu_i \mu_{C_i}(c)$$

- (c) Combine the inferences of all applicable rules in the rule base. Here, the *max* operation is used for “OR” (“ELSE”), to obtain the overall inference  $\hat{C}$ ; thus,

$$\mu_{\hat{C}}(c) = \max_i \mu_{\hat{C}_i}(c)$$

- (d) Defuzzify  $\hat{C}$  to obtain a crisp action  $\hat{c}$ , for instance using the centroid method. An alternative approach would be to first form the overall rule base  $R$  as:

$$\mu_R(e, c) = \min(\mu_{E_i}(e), \mu_{C_i}(c)) \text{ for rule } i$$

$$\mu_R(e, c) = \max_i \mu_{R_i}(e, c) \text{ for the complete rule base.}$$

Then, read off the value of  $\mu_R(e, c)$  corresponding to the point  $e = \hat{e}$ , to obtain the fuzzy inference  $\hat{C}$ ; thus,

$$\mu_{\hat{C}}(c) = \mu_R(\hat{e}, c)$$

which may be defuzzified as before.

Show that the two results are identical.

**Hint:** Use the *singleton function*  $s(x)$  defined as

$$\begin{aligned} s(e - \hat{e}) &= 1 \quad \text{for } e = \hat{e} \\ &= 0 \quad \text{elsewhere.} \end{aligned}$$

22. Knowledge-based decision-making is considered to involve cycles of perception, representation, and action. In fuzzy logic-based decision-making, indicate what each of these three terms represent.

23. Consider the rule (knowledge base):

“If an object is within  $3.0 \pm 0.2$  cm of the reference, then snap an image”

Suppose that, at a given instance, the data (context) shows that:

Position of the object = 2.9 cm

The inference of the knowledge-based system would be:

“The object is now ready for imaging”

Compare this with a fuzzy logic-based decision-making process where the rule base

$$R: A \rightarrow C$$

and the context

$$\hat{A} = \hat{a} \text{ with membership } \mu \text{ in } A$$

which result in an inference  $\hat{C}$  given by

$$\mu_{\hat{C}} = \min(\mu, \mu_R) \leq \mu$$

- 24.** Consider a rule base of the form

$$R: \bigcup_{i=1,m} E^i \rightarrow C^i$$

where there are  $m$  fuzzy levels (fuzzy resolution) for the condition variable  $E$ . Show that as  $m \rightarrow \infty$  for a fixed and finite support set for  $E$ , the rule base may be expressed as a crisp relation. Why is it that  $m$  cannot be very large in practice? How would an appropriate value be chosen for  $m$ ?

- 25.** In a fuzzy logic-based robot controller, it is attempted to improve the performance in the following manner:

By observing the change in performance due to a specific control action, the more effective rules are tagged and the associated improvement in performance is noted. Then, the rules are enhanced (e.g., by changing the membership functions; by adding a belief factor; by changing the scaling parameters) according to the required system behavior. Indicate difficulties in implementing such an approach.

- 26.** Intuitively, fuzzy logic is appropriate for inherently fuzzy problems rather than crisp ones. But we come across many applications where the original problem is crisp; and is made fuzzy for the purpose of using fuzzy logic techniques. Zadeh's *interpolative reasoning* is useful in these problems, as is the extension principle. In interpolative reasoning, a crisp relationship between two variables (say,  $u$  and  $v$ ) may be replaced by a set of fuzzy points on the variable plane. Each point, then, is a two-dimensional membership function (in  $u$  and  $v$ ). Discuss the use of interpolative reasoning in

- (a) an inherently fuzzy decision-making problem
- (b) an originally crisp problem.

Interpolative reasoning is widely used in fuzzy processor chips.

- 27.** Consider a knowledge-based hierarchical control system that has a “knowledge-based supervisor” which monitors and controls the hierarchies. The layers of the hierarchical structure are organized according to the *level of abstraction* of the processed information and the required speed of response.

- (a) Identify advantages of such a control structure.

Consider a feedback path from Level  $i$  to Level  $j$  such that  $j > i + 1$ . Denote this type of feedback as “Feedback Structure A.” Now, suppose that this feedback path is broken into two subpaths as follows:

Lower path: From  $i$  to  $m$ ;  $i < m < j$

Upper path: From  $m$  to  $j$

Denote this type of feedback as “Feedback Structure B.”

- (b) Discuss advantages and disadvantages of the two feedback structures.

28. Consider a control rule of the form:

If  $A$  and  $B$  and ... then  $C$

Suppose that the membership function  $\mu_C(c)$  of the control variable  $C$  is triangular with support interval  $[-d, d]$  and a unity peak at the mid-point of the support interval (i.e., at zero). For a given response condition (context), suppose that the combined membership grade of the context (say, using the *min* operation on the individual grades) is  $m$ . If the *dot* operation is used, instead of the *min* operation, for obtaining the membership of the corresponding control action  $C'$ , we have:

$$\mu_{C'}(c) = m\mu_C(c)$$

Obtain an expression for the fuzziness of  $C'$ , using the  $1/2$ -cut definition. Show that this fuzziness is highest at  $m = 1/\sqrt{2}$ .

29. Research shows that learning words and learning grammar are two quite different activities handled by different parts of the human brain. Can we consider this as a hierarchical problem where learning words is done at a lower level of the hierarchy, needing less intelligence than learning grammar?
30. (a) Explain what you mean by the stability of a fuzzy dynamic system. What approaches may be used to study the stability of a fuzzy dynamic system?  
 (b) Consider a first order fuzzy dynamic system whose free (unforced; input = 0) response is given by

$$X_{j+1} = X_j \circ R$$

where

$X_j$  = discrete membership function (column vector) of the  $j$ -th fuzzy state of the system.

$R$  = matrix representing the fuzzy rule base relation of state transition.

Investigate the stability of this decision-making process for the following examples:

$$(i) R = \begin{bmatrix} 0.1 & 0.4 \\ 0.8 & 1.0 \end{bmatrix}; \quad (ii) R = \begin{bmatrix} 0.1 & 0.3 \\ 0.2 & 0.4 \end{bmatrix}; \quad (iii) R = \begin{bmatrix} 0.1 & 0.4 \\ 0.3 & 0.2 \end{bmatrix}.$$

31. Fuzzy decision-making using just one condition variable is useful in fuzzy logic control, and particularly in the tuning of conventional controllers such as proportional-integral-derivative (PID) control.

Consider the case of one condition variable  $E$  and one action variable  $C$ . The rule base  $R$  is given by its membership function

$$\mu_R(e, c) = \min_i [\mu_{E^i}(e), \mu_{C^i}(c)]$$

where  $i$  denotes the rule number.

Now consider a crisp measurement  $\hat{e}$ . It may be fuzzified as  $E'$  whose membership function is

$$\mu_E(e) = \mu_{E'}(\hat{e})\delta(e - \hat{e})$$

where the delta function is defined as

$$\begin{aligned} \delta(e - \hat{e}) &= 1.0 \quad \text{for } e = \hat{e} \\ &= 0.0 \quad \text{elsewhere} \end{aligned}$$

Also, the membership function of  $E$  is given by

$$\mu_E(e) = \max_i \mu_{E^i}(e)$$

The sup-min composition is applied to determine the fuzzy decision  $C'$  corresponding to the fuzzy measurement  $E'$ .

Giving the necessary steps and explanations show that

$$\mu_{C'}(c) = \mu_R(\hat{e}, c)$$

32. (i) Prove that:

$$\min[x, \max(y, z)] = \max[\min(x, y), \min(x, z)]$$

**Hint:** First consider the case  $y > z$  and prove that LHS = RHS. Then switch  $y$  and  $z$ .

- (ii) Prove that:

$$\sup_a \max[f_1(a), f_2(a)] = \max[\sup_a f_1(a), \sup_a f_2(a)]$$

**Hint:** You may use a graphical approach. First consider the case

$$\sup_a f_1(a) > \sup_a f_2(a). \text{ Then switch } f_1 \text{ and } f_2.$$

- (iii) Consider two fuzzy rules  $R_1$  and  $R_2$  with membership functions  $\mu_{R_1}(a, c)$  and  $\mu_{R_2}(a, c)$  respectively. Note that the rules are of the form  $A \rightarrow C$  with  $a$  denoting the condition variable axis and  $c$  denoting the action variable axis. Now, with the “OR” connective between the rules, the rule base  $R$  formed by the two rules  $R_1$  and  $R_2$  is given by

$$\mu_R(a, c) = \mu_{R_1 \cup R_2}(a, c) = \max[\mu_{R_1}(a, c), \mu_{R_2}(a, c)]$$

Now consider a fuzzy measurement  $A'$  with membership function  $\mu_{A'}(a)$ .

Using the results obtained in parts (i) and (ii) show that the inference  $C'$  obtained by composing the measurement  $A'$  with the rule base obeys the relation

$$C' = A' \circ R = (A' \circ R_1) \cup (A' \circ R_2)$$

Specifically, using the results in parts (i) and (ii) show that

$$\sup_a \min[\mu_{A'}(a), \mu_R(a, c)] =$$

$$\max[\sup_a \min\{\mu_{A'}(a), \mu_{R_1}(a, c)\}, \sup_a \min\{\mu_{A'}(a), \mu_{R_2}(a, c)\}]$$

**Note:** In proving this result, you have proved that the composition operator “ $\circ$ ” distributes over the union operator “U”.

Give a practical use of your result in the context of fuzzy rule-based decision-making (for example, in fuzzy logic control).

1. De Silva, C.W. (1993) "Soft automation of industrial processes," *Engineering Applications of Artificial Intelligence*, vol. 6, no. 2, pp. 87–90.
2. De Silva, C.W., and MacFarlane, A.G.J. (1988) "Knowledge-based control structure for robotic manipulators," *Proc. IFAC Workshop on Artificial Intelligence in Real-Time Control*, Swansea, U.K., pp. 143–8, September.
3. De Silva, C.W., and MacFarlane, A.G.J. (1989) *Knowledge-Based Control with Application to Robots*, Springer-Verlag, Berlin.
4. De Silva, C.W., and MacFarlane, A.G.J. (1989) "Knowledge-based control approach for robotic manipulators," *International Journal of Control*, vol. 50, no. 1, pp. 249–73.
5. De Silva, C.W., and Lee, T.H. (1994) "Fuzzy logic in process control," *Measurements and Control*, vol. 28, no. 3, pp. 114–24, June.
6. De Silva, C.W. (1994) "A criterion for knowledge base decoupling in fuzzy logic control systems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24, no. 10, pp. 1548–52, October.
7. De Silva, C.W. (1995) *Intelligent Control: Fuzzy Logic Applications*, CRC Press, Boca Raton, FL.
8. De Silva, C.W. (1995) "Considerations of hierarchical fuzzy control," *Theoretical Aspects of Fuzzy Control*, H.T. Nguyen, M. Sugeno, R.M. Tong, and R.R. Yager (eds), J. Wiley & Sons, New York.
9. De Silva, C.W. (1997) "Intelligent control of robotic systems," *International Journal of Robotics and Autonomous Systems*, vol. 21, pp. 221–37.
10. De Silva, C.W. (ed.) (2000) *Intelligent Machines: Myths and Realities*, CRC Press, Boca Raton, FL.
11. Dubois, D., and Prade, H. (1980) *Fuzzy Sets and Systems*, Academic Press, Orlando.
12. Goulet, J.F., De Silva, C.W., Modi, V.J., and Misra, A.K. (2001) "Hierarchical control of a space-based deployable manipulator using fuzzy logic," *AIAA Journal of Guidance, Control, and Dynamics*, vol. 24, no. 2, pp. 395–405.
13. Jain, L.C., and De Silva, C.W. (eds) (1999) *Intelligent Adaptive Control: Industrial Applications*, CRC Press, Boca Raton, FL.
14. Mamdani, E.H. (1977) "Application of fuzzy logic to approximate reasoning using linguistic synthesis," *IEEE Trans. Comp.*, vol. C-26(12), pp. 1182–92.
15. Modi, V.J., Zhang, J., and De Silva, C.W. (in press) "Intelligent hierarchical modal control of a novel manipulator with slewing and deployable links," *Acta Astronautica*.
16. Procyk, T.J., and Mamdani, E.H. (1979) "A linguistic self-organizing controller," *Automatica*, vol. 15, pp. 15–30.
17. Rahbari, R., and De Silva, C.W. (2001) "Comparison of two inference methods for P-type fuzzy logic control through experimental investigation using a hydraulic manipulator," *International Journal Engineering Applications of Artificial Intelligence*, vol. 14, no. 6, pp. 763–84.
18. Sugeno, M. (ed.) (1985) *Industrial Applications of Fuzzy Control*, North-Holland, Amsterdam.
19. Tang, P.L., Poo, A.N., and De Silva, C.W. (2001) "Knowledge-based extension of model-referenced adaptive control with application to an industrial process," *Journal of Intelligent and Fuzzy Systems*, vol. 10, no. 3, pp. 159–83.

20. Wu, Q.M., and De Silva, C.W. (1996) "Dynamic switching of fuzzy resolution in knowledge-based self-tuning control," *Journal of Intelligent and Fuzzy Systems*, vol. 4, no. 1, pp. 75–87.
21. Yan, G.K.C., De Silva, C.W., and Wang, G.X. (2001) "Experimental modeling and intelligent control of a wood-drying kiln," *International Journal of Adaptive Control and Signal Processing*, vol. 15, pp. 787–814.
22. Zadeh, L.A. (1975) "The concept of a linguistic variable and its application to approximate reasoning," Parts 1–3, *Information Science*, pp. 43–80, 199–249, 301–57.

# Connectionist modeling and neural networks



# Fundamentals of artificial neural networks

## Chapter outline

4.1	Introduction	223
4.2	Learning and acquisition of knowledge	224
4.3	Features of artificial neural networks	226
4.4	Fundamentals of connectionist modeling	233
4.5	Summary	246
	Problems	246
	References	248

## 4.1 Introduction

The immense capabilities of the human brain in processing information and making instantaneous decisions, even under very complex circumstances and under uncertain environments, have inspired researchers in studying and possibly mimicking the computational abilities of this wonder. What a human can achieve in a very short time, for instance, in terms of pattern recognition and obstacle avoidance within an unknown environment, would take very expensive computer resources (programmers, training experts, expensive hardware) and much longer to get comparable results. This is mainly due to the way humans process information. Indeed, researchers have shown for many years that brains make computations in a radically different manner to that done by digital computers. Unlike computers, which are programmed to solve problems using sequential algorithms, the brain makes use of a massive network of parallel and distributed computational elements called neurons. The large number of connections linking these elements provides humans with the very powerful capability of learning. Motivated by this very efficient computational biological model, scientists have for the last few decades attempted to build computational systems, which can process information in a similar way. Such systems are called artificial neural networks (ANN) or connectionist models. They are composed of a large number of highly interconnected

processing elements analogous in functionality to biological neurons and are tied together with weighted connections corresponding to brain synapses.

## 4.2 Learning and acquisition of knowledge

Learning is accomplished in general by developing algorithms that allow the system to learn for itself from a set of input/output training data. One major goal of learning algorithms is to combine the main features of a computing machine with those of human expertise to infer as many correct decisions as possible. An increasing number of systems are being designed today to have the very distinctive feature of learning. This is done by adjusting their parameters in response to unpredictable changes in their dynamics or their operating environment without the need for an explicit knowledge of a system model or rules that guide its behavior.

Once a learning capability has been implemented within a given system, it continues acquiring knowledge and uses the stored information to draw right decisions for similar future situations. Two well-known approaches for designing learning systems employ symbolic-learning and numerical-learning-based techniques. Although the focus here is on numerical learning, it is nevertheless interesting to preview as well the main features of symbolic learning.

### 4.2.1 Symbolic learning

Symbolic learning pertains to the ability of a given system in formulating and updating its knowledge base from a set of well-structured feedback data related to the performance of the system. Systems with symbolic learning capabilities can update their operating set of rules in response to changes in the system dynamics or its operating environment. This depends to a large extent on the degree of supervision to which the system may be subjected. Several categories of symbolic learning have been proposed in the literature [1]. These include rote learning, explorational learning, inductive learning, and explanation-based learning. What distinguishes these learning algorithms from one another is essentially the level of supervision. Specifically, the inductive learning approach has received more attention due to its relative ease of implementation and to the rich feedback information it generates.

- (1) Learning by induction involves the synthesis of inference rules from a set of specific example solutions.
- (2) Self-organizing fuzzy logic systems represent a typical example of inductive learning-based tools.

### 4.2.2 Numerical learning

In recent years, a large body of research on numerical learning has focused specifically on artificial neural networks\* (ANN). This is an area of research

---

\* Artificial neural networks and neural networks are synonymous in the context of this book.

that has seen a renewed interest, following an uncertain start in the 1960s and the early 1970s. Numerical-learning-based algorithms represent a class of very useful learning tools used most often for the design of neural networks. They provide a given network with the capacity of adjusting its parameters in response to training signals. In some cases, learning is acquired by practice through training (supervised learning). In others the system is presented with a number of patterns and it is up to the network, through well-defined guidelines, to group these patterns into categories (unsupervised learning). For instance, in the case of supervised learning, the system is presented with an *a priori* known set of input/output data. The learning mechanism is carried out through an optimization process, during which the system attempts to approximate the desired response vector with the output of the learning algorithm. This is done by solving an optimization problem involving  $m$  patterns of desired responses  $\mathbf{t}^{(1)}, \mathbf{t}^{(2)}, \dots, \mathbf{t}^{(m)}$ , and their corresponding inputs  $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(m)}$ , and a set of *a priori* unknown optimization parameters called connection weights  $\mathbf{W} = [w_1, w_2, \dots, w_p]$ . The superscript  $m$  denotes the number of training patterns presented to the network and the subscript  $p$  represents the number of interconnection weights among all nodes of the network. Each training input vector  $\mathbf{x}$  is composed of  $l$  components  $x_i$ , while each training output vector  $\mathbf{t}$  is composed of  $q$  components  $t_q$ . The optimization algorithm seeks the minimization of the cumulative sum of the errors  $e$  between the  $k$ -th desired output vector  $\mathbf{t}^{(k)}$  and the  $k$ -th output vector  $\mathbf{o}^k$  given by  $h(\mathbf{W}, \mathbf{x}^{(k)})$ , corresponding to the actual output of the network in response to the excitation  $\mathbf{x}^{(k)}$ , shown in Figure 4.1. The mapping  $h$  represents here the input–output representation of the neural network. The optimization problem is now expressed as

$$\min_{\mathbf{w}} \sum_{k=1}^m e(\mathbf{o}^{(k)}, \mathbf{t}^{(k)}) = \min_{\mathbf{w}} \sum_{k=1}^m e(h(\mathbf{w}, \mathbf{x}^{(k)}), \mathbf{t}^{(k)}) \quad (4.1)$$

The training algorithm keeps adjusting the weights according to well-defined learning rules. As more data become available, the learning system

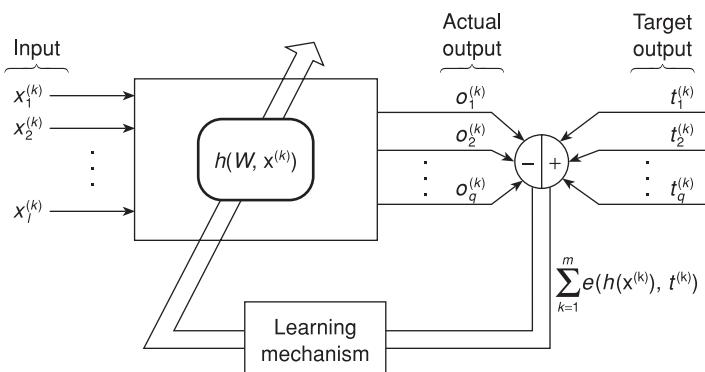


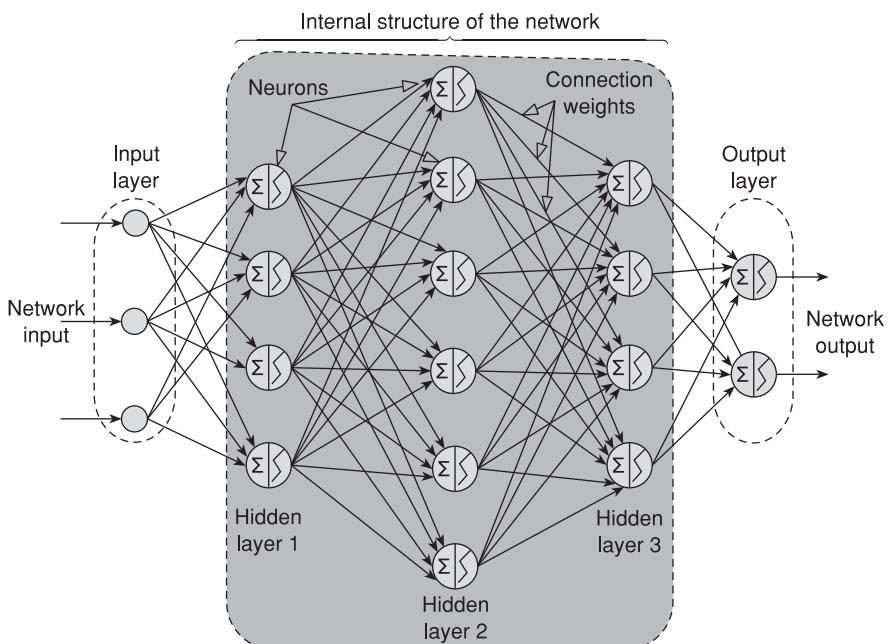
Figure 4.1: Schematic representation of a supervised ANN

continues updating the weights to comply with the optimization criteria while adapting to new situations. Several techniques to solve this optimization problem have been proposed in the literature [2]. Some of the well-known approaches include genetic algorithms, annealing algorithms, and gradient-descent-based algorithms. Details on the characteristics of these algorithms and others can be found in [2]. Once a given algorithm has converged, the weights become a means for memory storage, hence providing the system with the capability of tackling a given problem through the learning it has acquired during the training stage.

The two major numerical learning algorithms most often used for the design of neural networks are the supervised and the unsupervised algorithms. Details on these algorithms and others along with an outline of their main features are provided in subsequent sections and chapters.

### 4.3 Features of artificial neural networks

As mentioned previously, an artificial neural network (ANN) is typically composed of a set of parallel and distributed processing units, called nodes or neurons. These are usually ordered into layers, appropriately interconnected by means of unidirectional (or bi-directional in some cases) weighted signal channels, called connections or synaptic weights, as shown in Figure 4.2.



**Figure 4.2:** Typical representation of a feedforward (unidirectional) artificial neural network with three inputs and two outputs

The internal architecture of ANN provides powerful computational capabilities, allowing for the simultaneous exploration of different competing hypotheses. Massive parallelism and computationally intensive learning through examples make them suitable for application in nonlinear functional mapping, speech and pattern recognition, categorization, data compression, and many other applications characterized by complex dynamics and possibly uncertain behavior. Neural networks gather their knowledge through detection of patterns and relationships found in the data provided to them. Three important features generally characterize an artificial neural network: the network topology, the network transfer functions, and the network learning algorithm.

### 4.3.1 Neural network topologies

These correspond to the ordering and organization of the nodes from the input layer to the output layer of the network. In fact, the way the nodes and the interconnections are arranged within the layers of a given ANN determines its topology. The choice for using a given topology is mainly dictated by the type of problem being considered. Some neural networks designers classify ANN according to how the nodes are organized and hence how data is processed through the network. The two well-known ANN topologies are the feedforward and the recurrent architectures.

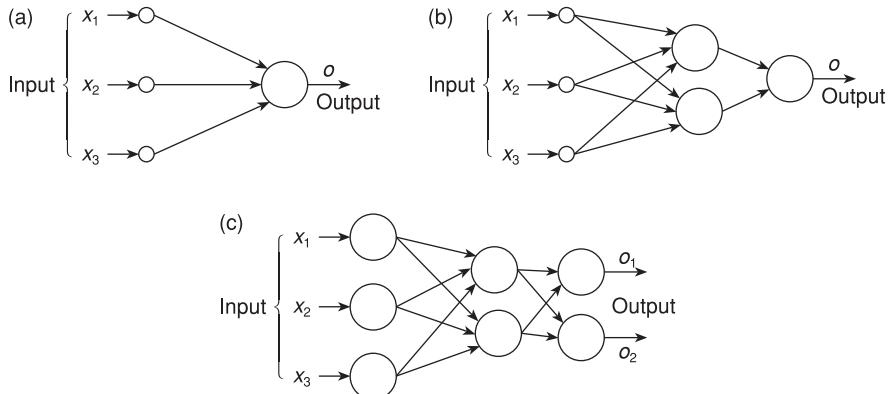
#### 4.3.1.1 *The feedforward topology*

A network with a feedforward (FF) architecture has its nodes hierarchically arranged in layers starting with the input layer and ending with the output layer. In between, a number of internal layers, also called hidden layers, provide most of the network computational power. The nodes in each layer are connected to the next layer through unidirectional paths starting from one layer (source) and ending at the subsequent layer (sink). This means that the outputs of a given layer feed the nodes of the following layer in a forward path as shown in Figure 4.3. Because of their structure, such networks are termed as feedforward networks. They are also occasionally called open-loop networks given the absence of feedback flow of information in their structure.

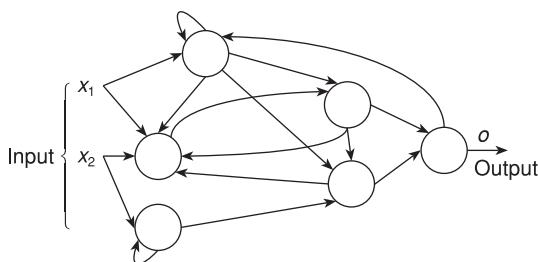
The feedforward topology has been very popular due to its association with a quite powerful and relatively robust learning algorithm called the backpropagation learning algorithm (BPL). The multilayer perceptron network and the radial basis function network are among the well-known networks using the feedforward topology.

#### 4.3.1.2 *The recurrent topology*

Unlike feedforward networks, recurrent networks (RN) allow for feedback connections among their nodes as illustrated in Figure 4.4. They are structured in such a way as to permit storage of information in their output nodes through dynamic states, hence providing the network with some sort of “memory.” While FF networks map input into output and are static in the



**Figure 4.3:** A typical structure of neural networks with feedforward topology, (a) multi-input, single output, no hidden layer, (b) multi-input, single output, one hidden layer, (c) multi-input, multi-output, one hidden layer

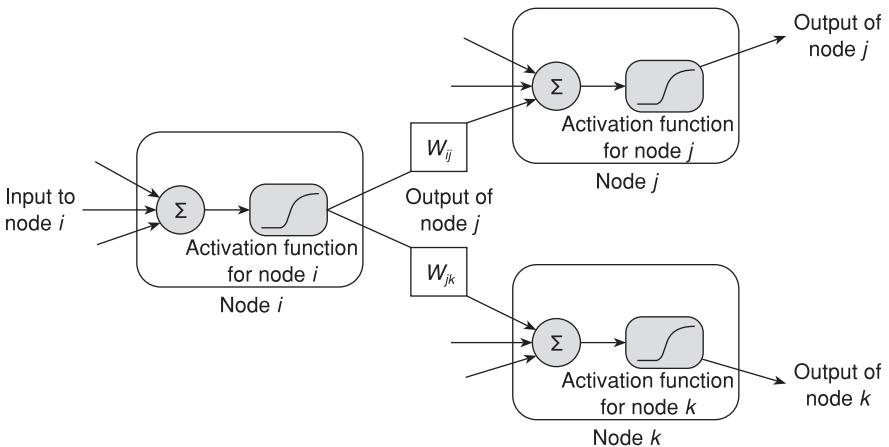


**Figure 4.4:** A typical neural network with recurrent topology

sense that the output of a given pattern of inputs is independent of the previous state of the network, recurrent networks map states into states and as such are very useful for modeling and identifying dynamic systems. This means that the feedback available for a given input node allows it to pass to another state as soon as an output has been delivered at the other end of the network. Several well-known neural networks have been designed based on the recurrent topology. Such networks include the Hopfield network and the time delayed neural networks (TDNN).

#### 4.3.2 Neural network activation functions

The basic elements of the computational engine for a neural network are the neurons. These are sorts of simple processors which take the weighted sum of their inputs from other nodes and apply to them a nonlinear mapping (not necessarily linear) called an activation function before delivering the output to the next neuron (Figure 4.5). The output  $o_k$  of a typical neuron ( $k$ ) having ( $l$ ) inputs is given as:



**Figure 4.5:** Interconnections between neurons

$$o_k = f \left( \sum_l w_{ik} x_i - \theta_k \right) \quad (4.2)$$

where  $f$  is the node's activation function,  $x_1, x_2, \dots, x_l$  are the node's inputs,  $w_{1k}, w_{2k}, \dots, w_{lk}$  are the connections weights, and  $\theta_k$  is the node's threshold. The processing activity within a given layer is done simultaneously, hence providing the neural network with the powerful capability of parallel computing. The bias effect (threshold value) is intended to occasionally inhibit the activity of some nodes. As neural networks may vary in terms of their structure as described previously, they may as well vary in terms of their activation function.

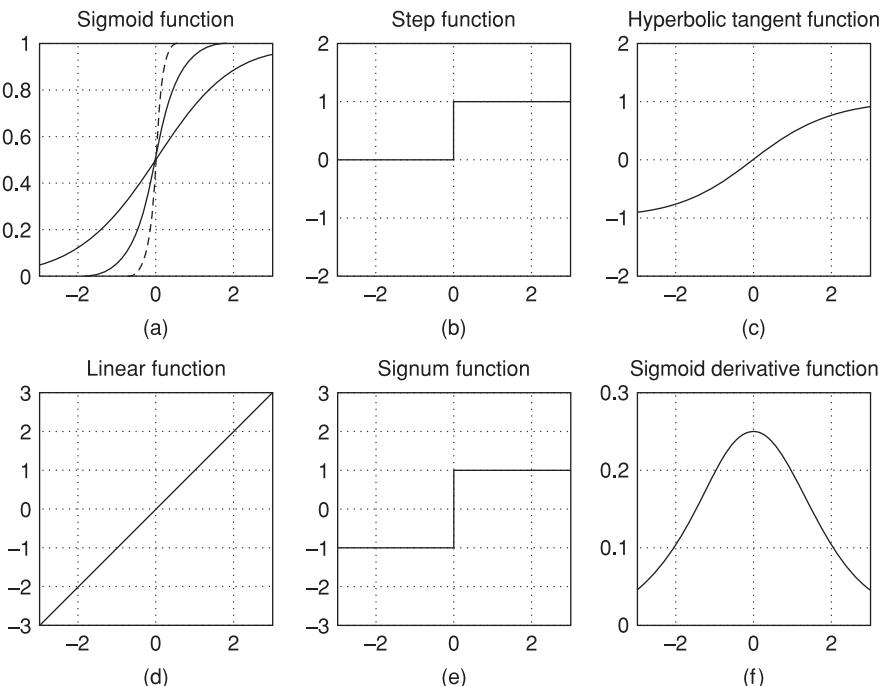
Depending on the problem at hand and on the location of the node within a given layer, the activation functions can take different forms: sigmoid mapping, signum function, step function or linear correspondence. The mathematical representation for some of these mappings is given below:

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)} \quad (4.3)$$

$$\text{signum}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases} \quad (4.4)$$

$$\text{step}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \quad (4.5)$$

Figure 4.6 illustrates the profile of six activation functions commonly used in implementations of neural networks.



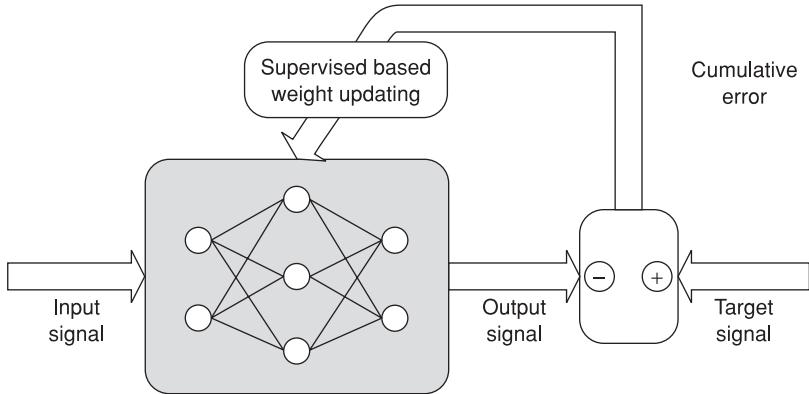
**Figure 4.6:** Typical profiles of six activation functions

### 4.3.3 Neural network learning algorithms

Learning algorithms are used to update the weight parameters at the interconnection level of the neurons during the training process of the network. While some designers have classified neural networks according to topologies or architectures, others have classified them according to the learning algorithm used by the network. The three well-known and most often used learning mechanisms are the supervised, the unsupervised (or self-organized) and the reinforced. A brief overview of each one of these weight-updating mechanisms is provided next. For an in-depth study of learning algorithms, the reader may wish to consult [3].

#### 4.3.3.1 Supervised learning

The main feature of the supervised (or active) learning mechanism (see Figure 4.7) is the training by examples. This means that an external teacher provides the network with a set of input stimuli for which the output is *a priori* known. During the training process, the output results are continuously compared with the desired data. An appropriate learning rule (such as the gradient descent rule) uses the error between the actual output and the target data to adjust the connection weights so as to obtain, after a number of iterations, the closest match between the target output and the actual output. Supervised learning is particularly useful for feedforward networks. A

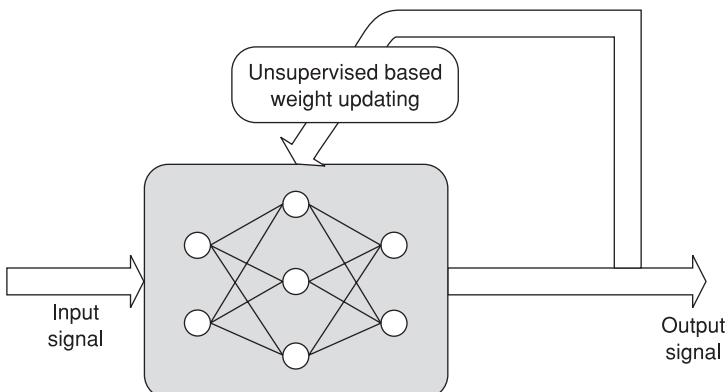


**Figure 4.7:** Schematic representation of supervised learning

number of supervised learning algorithms have been suggested in the literature. The backpropagation algorithm, first developed by Werbos in 1974 [4], which is also based on the gradient descent optimization technique and the least mean square algorithm (Widrow and Hoff, 1960)[5], are among the most commonly used supervised learning rules. More on these algorithms and optimization rules is included in subsequent chapters.

#### 4.3.3.2 Unsupervised learning

Unlike supervised learning, unsupervised or self-organized learning (Figure 4.8) does not involve an external teacher and relies instead upon local information and internal control. The training data and input patterns are presented to the system, and through predefined guidelines, the system discovers emergent collective properties and organizes the data into clusters or categories. Because of the way the network adjusts its connection weights in response to the presented input data, unsupervised learning algorithms have been known

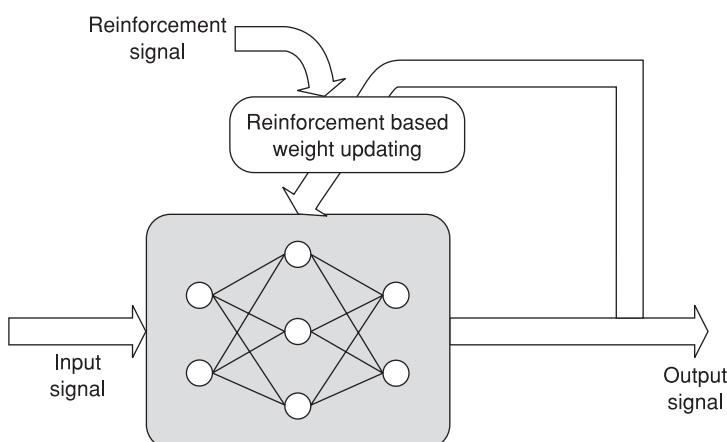


**Figure 4.8:** Schematic representation of unsupervised learning

as open-loop adaptation learning schemes. An unsupervised learning scheme operates as follows. A set of training data is presented to the system at the input layer level. The network connection weights are then adjusted through some sort of competition among the nodes of the output layer, where the successful candidate will be the node with the highest value. In the process, the algorithm strengthens the connection between the incoming pattern at the input layer and the node output corresponding to the winning candidate. In addition to the strengthening of the connections between the input layer and the winning output node, the unsupervised learning scheme may be used for adjusting the weights of the connections leading to the neighboring nodes at the output layer. This is controlled by what is called the neighborliness parameter, and it has the major property of making groups of output nodes behave as single entities with particular features. More on the self-organizing mechanisms will be provided in a subsequent chapter.

#### 4.3.3.3 Reinforcement learning

Reinforcement learning (Figure 4.9), also known as graded learning, has been receiving an increased interest lately because of its many attractive learning features, which mimic in a way the adjusting behavior of humans when interacting with a given physical environment. This is another type of learning mechanism by means of which the network connections are modified according to feedback information provided to the network by its environment. This information simply instructs the system on whether or not a correct response has been obtained. In the case of a correct response, the corresponding connections leading to that output are strengthened otherwise they are weakened. This type of learning strategy based on the reward/punishment paradigm has several similarities with the biological learning system. Unlike supervised learning, reinforcement learning (RL) doesn't get information on what the output should be when the network is presented with a given input



**Figure 4.9:** Schematic representation of reinforcement learning

pattern. Reinforcement learning does also differ from unsupervised learning (UL) in that UL doesn't provide the network with information on whether the output is correct or not, but rather operates on the premises of finding pattern regularity among the exemplars presented to the network. Given the nature of this learning scheme, random search strategies have been used to attain the correct output every time the system is presented with an excitation from its environment. This explorational aspect of learning ultimately leads to an improved capability of the system to deliver the expected output every time the system is presented with an input pattern. Among the strategies used to implement a reinforced learning algorithm are the reinforcement comparison, the adaptive heuristic critic, the Q learning, and the policy only scheme [2].

## 4.4 Fundamentals of connectionist modeling

Researchers studying neural networks have come a long way since the early days of connectionist research, which started in the mid-1940s. The path towards establishing connectionist-based computation, as a viable tool for solving a wide range of complex problems, has not always been a smooth one. A number of attempts with mixed results have been made in the past 40 years. But only quite recently (in the mid-1980s) has this area of research seen a wider acceptability among researchers and practitioners. This is mostly due to the discovery of powerful new learning techniques and more robust architectures that ultimately led to successful implementation of neural networks for a large number of applications ranging from communication systems to data mining for marketing and sales. These systems usually involve complex dynamics and large amounts of data that may have been corrupted with noise. Before getting into a more detailed description of a number of well-known neural networks and their wide range of applications in the industry, it would be informative to the reader to have an overview of the earliest attempts at constructing connectionist models. These same models are considered by many as the backbone of many of today's powerful and versatile neural network structures.

### 4.4.1 McCulloch–Pitts models

The McCulloch–Pitts neuron model designed by W.S. McCulloch and W. Pitts in the early 1940s [6] is considered by many as the first serious attempt to model the computing process of the biological neuron. Although this model was very limited in terms of computing capability and doesn't actually provide learning, it nevertheless sparked a wave of research interest in the field that ultimately led to the design of more reliable and robust connectionist models.

This neuron model is quite simple in design as shown in Figure 4.10. It collects the incoming signals  $x_1, x_2, \dots, x_l$ , multiplies them by corresponding weights  $w_1, w_2, \dots, w_l$  and compares the result with a predetermined bias  $\theta$ .

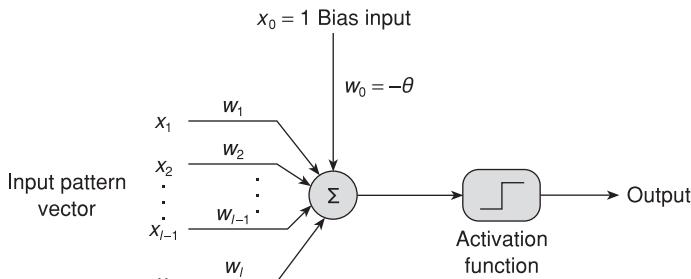


Figure 4.10: McCulloch–Pitts neuron

before applying some sort of nonlinear (not necessarily linear) activation mapping  $f$  resulting in the output  $o$  with expression given by:

$$o = f\left(\sum_{i=1}^l w_i x_i - \theta\right) \quad (4.6)$$

If the output value of the combiner ( $\sum w_i x_i$ ), which is also the weighted sum of the different signal inputs, is larger than the bias  $\theta$ , then an output value of 1 is generated, otherwise the result is zero. This is done through the activation function  $f(\cdot)$ , which corresponds here to the step function defined earlier. Usually, and for simplicity of notation, the bias  $\theta$  is expressed as another input to the neuron with value  $x_0 = 1$  and a connection weight  $w_0 = -\theta$ . As such the output  $o$  becomes expressed as:

$$o = f\left(\sum_{i=0}^l w_i x_i\right) \quad (4.7)$$

Notice here the absence of any type of learning since there is no updating mechanism for the synaptic weights once the system has been presented with a set of training input–output data.

#### 4.4.2 Perceptron

The McCulloch–Pitts model is quite intuitive in concept, yet it has severe limitations, particularly in terms of lack of learning capabilities. This is primarily due to the fact that the model topology is based on a fixed set of weights and thresholds. To overcome this severe shortcoming, other models involving learning capabilities have been suggested on the basis they would have to adjust the connection weights in accordance with some sort of optimization mechanism. In the early 1960s, Rosenblatt of Cornell University [7] developed a trainable model of an artificial neuron using a supervised learning procedure in which the system adjusts its weights in response to a comparative signal computed between the actual output and the target output. The Rosenblatt

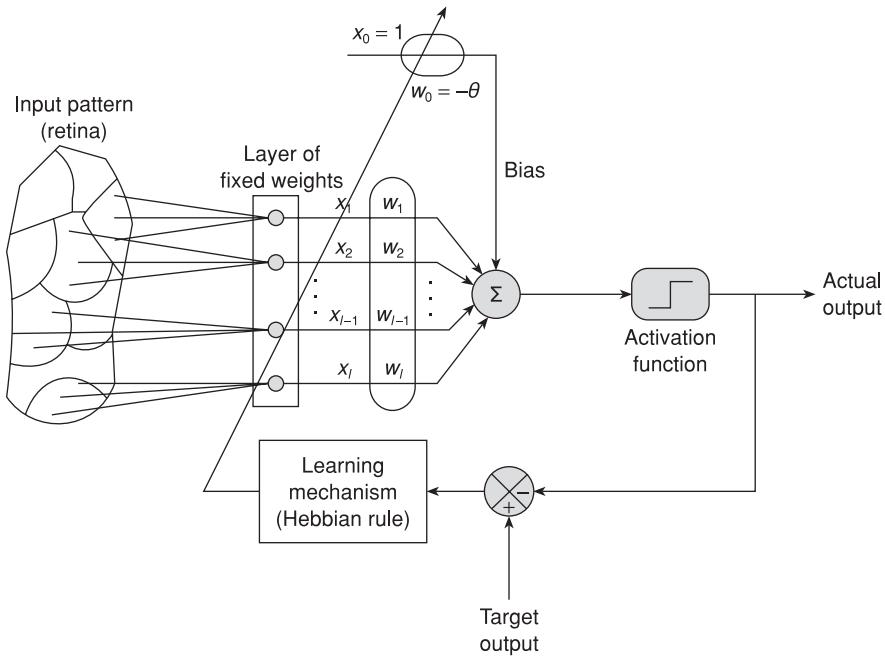


Figure 4.11: Representation of the perceptron

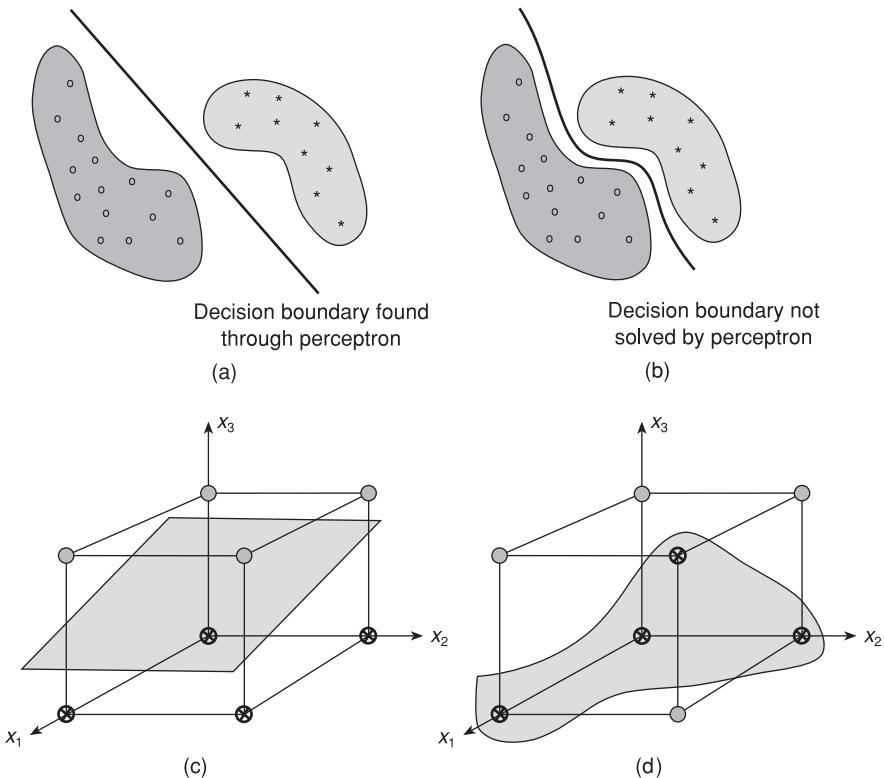
model [7], also called the *perceptron*, was designed for the purpose of pattern classification of linearly separable sets [3]. By definition, sets are linearly separable if there exists a hyperplanar multidimensional decision boundary that classifies the data input into two classes.

In terms of architecture, the *perceptron* is composed of a hierarchical three-level structure. The topology of the perceptron includes an input level corresponding to the sensory unit or retina; a second-level unit, also called a feature detector unit, involving nodes connected to the input but with fixed connection weights and thresholds; and a third-level unit involving the output layer composed of one single node but with adjustable connection weights. This structure is illustrated in Figure 4.11.

The activation function used by Rosenblatt in the perceptron is the step or the hard limiting activation function, and the learning algorithm used to adjust the weights is the perceptron learning rule. It was shown (perceptron convergence theorem [3]) that as long as the patterns used to train the perceptron are linearly separable, the learning algorithm should converge in a finite number of steps. This means that a decision boundary should be obtained to classify the patterns (Figure 4.12).

To illustrate the ideas, and using the same notation as earlier, the boundary line (or hyperplane) takes the form of:

$$\sum_i w_i x_i - \theta = 0$$



**Figure 4.12:** (a) linearly vs. (b) nonlinearly separable patterns in 2D space  
 (c) linearly vs. (d) nonlinearly separable patterns in 3D space

In the two-dimensional case, this translates into finding the line given by  $w_1x_1 + w_2x_2 - \theta = 0$ , which after learning should adequately classify the patterns as shown in Figure 4.13. As for the steps used to train a given perceptron, they are outlined as follows:

**Step 1:** Initialize weights and thresholds to small random values.

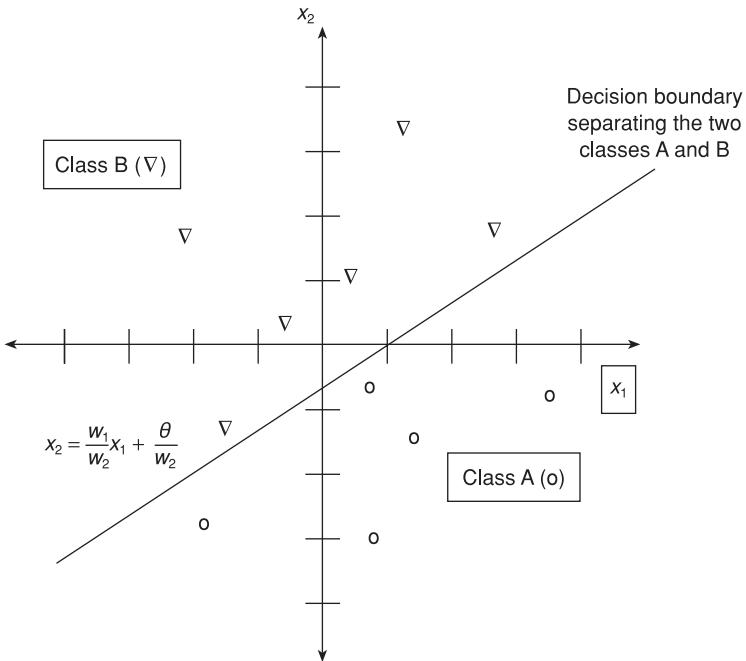
**Step 2:** Choose an input–output pattern from the training input–output data set  $(\mathbf{x}^{(k)}, t^{(k)})$ .

**Step 3:** Compute the actual output  $o = f\left(\sum_{i=1}^l w_i x_i - \theta\right)$ .

**Step 4:** Adjust the weights according to the perceptron learning rule:

$$\Delta w_i = \eta \left[ t - f\left(\sum_i w_i x_i - \theta\right) \right] x_i$$

For the case where  $f$  is the signum function (output of the perceptron is either 1 or -1), the connection update rule becomes expressed as:



**Figure 4.13:** Perceptron decision boundary adequately classifying linearly separable patterns

$$\Delta w_i = \begin{cases} 2\eta t x_i & \text{if } t \neq \operatorname{sgn}\left(\sum_i w_i x_i - \theta\right) \\ 0 & \text{otherwise} \end{cases}$$

and

$$\Delta \theta = \begin{cases} -2\eta t & \text{if } t \neq \operatorname{sgn}\left(\sum_i w_i x_i - \theta\right) \\ 0 & \text{otherwise} \end{cases}$$

with  $\eta$  being a positive update rate ranging from 0 to 1 representing the learning rate.

**Step 5:** In case the weights do not reach steady-state values  $\Delta w_i = 0$ , repeat starting from step 2 and choose another training pattern. This is known as epoch training. This learning procedure is very similar to the *Hebbian learning rule* [3], with the difference that the connection weights here are not updated if the network responds correctly.

### Example 4.1

We need to train the network using the following set of input and desired output training vectors:

$$\begin{aligned}(\mathbf{x}^{(1)} &= [1, -2, 0, -1]^T; t^{(1)} = -1), \\ (\mathbf{x}^{(2)} &= [0, 1.5, -0.5, -1]^T; t^{(2)} = -1), \\ (\mathbf{x}^{(3)} &= [-1, 1, 0.5, -1]^T; t^{(3)} = +1),\end{aligned}$$

with initial weight vector  $\mathbf{w}^{(1)} = [1, -1, 0, 0.5]^T$ , and the learning rate  $\eta = 0.1$ .

#### Epoch 1

Introducing the first input vector  $\mathbf{x}^{(1)}$  to the network, we get:

$$o^{(1)} = \text{sgn}(\mathbf{w}^{(1)^T} \mathbf{x}^{(1)}) = \text{sgn} \left( [1, -1, 0, 0.5] \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} \right) = +1 \neq t^{(1)},$$

then

$$\begin{aligned}\mathbf{w}^{(2)} &= \mathbf{w}^{(1)} + \eta[t^{(1)} - o^{(1)}]\mathbf{x}^{(1)} \\ &= \mathbf{w}^{(1)} + 0.1(-2)\mathbf{x}^{(1)} \\ &= [0.8, -0.6, 0, 0.7]^T\end{aligned}$$

Introducing the second input vector  $\mathbf{x}^{(2)}$  to the network, we get:

$$o^{(2)} = \text{sgn}(\mathbf{w}^{(2)^T} \mathbf{x}^{(2)}) = \text{sgn} \left( [0.8, -0.6, 0, 0.7] \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix} \right) = -1 = t^{(2)},$$

then

$$\mathbf{w}^{(3)} = \mathbf{w}^{(2)}$$

Introducing the third input vector  $\mathbf{x}^{(3)}$  to the network, we get:

$$o^{(3)} = \text{sgn}(\mathbf{w}^{(3)^T} \mathbf{x}^{(3)}) = \text{sgn} \left( [0.8, -0.6, 0, 0.7] \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix} \right) = -1 \neq t^{(3)},$$

then

$$\begin{aligned}\mathbf{w}^{(4)} &= \mathbf{w}^{(3)} + \eta[t^{(3)} - o^{(3)}]\mathbf{x}^{(3)} \\ &= \mathbf{w}^{(3)} + 0.1(2)\mathbf{x}^{(3)} \\ &= [0.6, -0.4, 0.1, 0.5]^T\end{aligned}$$

### Epoch 2

We reuse here the training set  $(\mathbf{x}^{(1)}, t^{(1)})$ ,  $(\mathbf{x}^{(2)}, t^{(2)})$  and  $(\mathbf{x}^{(3)}, t^{(3)})$  as  $(\mathbf{x}^{(4)}, t^{(4)})$ ,  $(\mathbf{x}^{(5)}, t^{(5)})$  and  $(\mathbf{x}^{(6)}, t^{(6)})$ , respectively.

Introducing the first input vector  $\mathbf{x}^{(4)}$  to the network, we get:

$$o^{(4)} = \text{sgn}(\mathbf{w}^{(4)T}\mathbf{x}^{(4)}) = \text{sgn} \left( [0.6, -0.4, 0.1, 0.5] \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} \right) = +1 \neq t^{(4)},$$

then

$$\begin{aligned}\mathbf{w}^{(5)} &= \mathbf{w}^{(4)} + \eta[t^{(4)} - o^{(4)}]\mathbf{x}^{(4)} \\ &= \mathbf{w}^{(4)} + 0.1(-2)\mathbf{x}^{(4)} \\ &= [0.4, 0, 0.1, 0.7]^T\end{aligned}$$

Introducing the second input vector  $\mathbf{x}^{(5)}$  to the network, we get:

$$o^{(5)} = \text{sgn}(\mathbf{w}^{(5)T}\mathbf{x}^{(5)}) = \text{sgn} \left( [0.4, 0, 0.1, 0.7] \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix} \right) = -1 = t^{(5)},$$

then

$$\mathbf{w}^{(6)} = \mathbf{w}^{(5)}$$

Introducing the third input vector  $\mathbf{x}^{(6)}$  to the network, we get:

$$o^{(6)} = \text{sgn}(\mathbf{w}^{(6)T}\mathbf{x}^{(6)}) = \text{sgn} \left( [0.4, 0, 0.1, 0.7] \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix} \right) = -1 \neq t^{(6)},$$

then

$$\begin{aligned}\mathbf{w}^{(7)} &= \mathbf{w}^{(6)} + \eta[t^{(6)} - o^{(6)}]\mathbf{x}^{(6)} \\ &= \mathbf{w}^{(6)} + 0.1(2)\mathbf{x}^{(6)} \\ &= [0.2, 0.2, 0.2, 0.5]^T\end{aligned}$$

### Epoch 3

We reuse now the training set  $(\mathbf{x}^{(1)}, t^{(1)})$ ,  $(\mathbf{x}^{(2)}, t^{(2)})$  and  $(\mathbf{x}^{(3)}, t^{(3)})$  as  $(\mathbf{x}^{(7)}, t^{(7)})$ ,  $(\mathbf{x}^{(8)}, t^{(8)})$  and  $(\mathbf{x}^{(9)}, t^{(9)})$ , respectively.

Introducing the first input vector  $\mathbf{x}^{(7)}$  to the network, we get:

$$o^{(7)} = \text{sgn}(\mathbf{w}^{(7)T} \mathbf{x}^{(7)}) = -1 = t^{(2)},$$

then

$$\mathbf{w}^{(8)} = \mathbf{w}^{(7)}$$

Introducing the second input vector  $\mathbf{x}^{(8)}$  to the network, we get:

$$y^{(8)} = \text{sgn}(\mathbf{w}^{(8)T} \mathbf{x}^{(8)}) = -1 = t^{(8)},$$

then

$$\mathbf{w}^{(9)} = \mathbf{w}^{(8)}$$

Introducing the third input vector  $\mathbf{x}^{(9)}$  to the network, we get:

$$o^{(9)} = \text{sgn}(\mathbf{w}^{(9)T} \mathbf{x}^{(9)}) = -1 \neq t^{(9)},$$

then

$$\begin{aligned}\mathbf{w}^{(10)} &= \mathbf{w}^{(9)} + \eta[t^{(9)} - o^{(9)}]\mathbf{x}^{(9)} \\ &= \mathbf{w}^{(9)} + 0.1(2)\mathbf{x}^{(9)} \\ &= [0, 0.4, 0.3, 0.3]^T\end{aligned}$$

### Epoch 4

We reuse the training set  $(\mathbf{x}^{(1)}, t^{(1)})$ ,  $(\mathbf{x}^{(2)}, t^{(2)})$  and  $(\mathbf{x}^{(3)}, t^{(3)})$  as  $(\mathbf{x}^{(10)}, t^{(10)})$ ,  $(\mathbf{x}^{(11)}, t^{(11)})$  and  $(\mathbf{x}^{(12)}, t^{(12)})$ , respectively.

Introducing the first input vector  $\mathbf{x}^{(10)}$  to the network, we get:

$$o^{(10)} = \text{sgn}(\mathbf{w}^{(10)T} \mathbf{x}^{(10)}) = -1 = t^{(10)},$$

then

$$\mathbf{w}^{(11)} = \mathbf{w}^{(10)}$$

Introducing the second input vector  $\mathbf{x}^{(11)}$  to the network, we get:

$$o^{(11)} = \text{sgn}(\mathbf{w}^{(11)\top} \mathbf{x}^{(11)}) = +1 \neq t^{(11)},$$

then

$$\begin{aligned}\mathbf{w}^{(12)} &= \mathbf{w}^{(11)} + \eta[t^{(11)} - o^{(11)}]\mathbf{x}^{(11)} \\ &= \mathbf{w}^{(11)} + 0.1(-2)\mathbf{x}^{(11)} \\ &= [0, 0.1, 0.4, 0.5]^\top\end{aligned}$$

Introducing the third input vector  $\mathbf{x}^{(12)}$  to the network, we get:

$$o^{(12)} = \text{sgn}(\mathbf{w}^{(12)\top} \mathbf{x}^{(12)}) = -1 \neq t^{(12)},$$

then

$$\begin{aligned}\mathbf{w}^{(13)} &= \mathbf{w}^{(12)} + \eta[t^{(12)} - o^{(12)}]\mathbf{x}^{(12)} \\ &= \mathbf{w}^{(12)} + 0.1(2)\mathbf{x}^{(12)} \\ &= [-0.2, 0.3, 0.5, 0.3]^\top\end{aligned}$$

Introducing the input vectors for another epoch will result in no change to the weights which indicates that  $\mathbf{w}^{(13)}$  is the solution for this problem; that is

$$w_1 = -0.2, w_2 = 0.3, w_3 = 0.5, \text{ and } w_4 = 0.3.$$

### Example 4.2

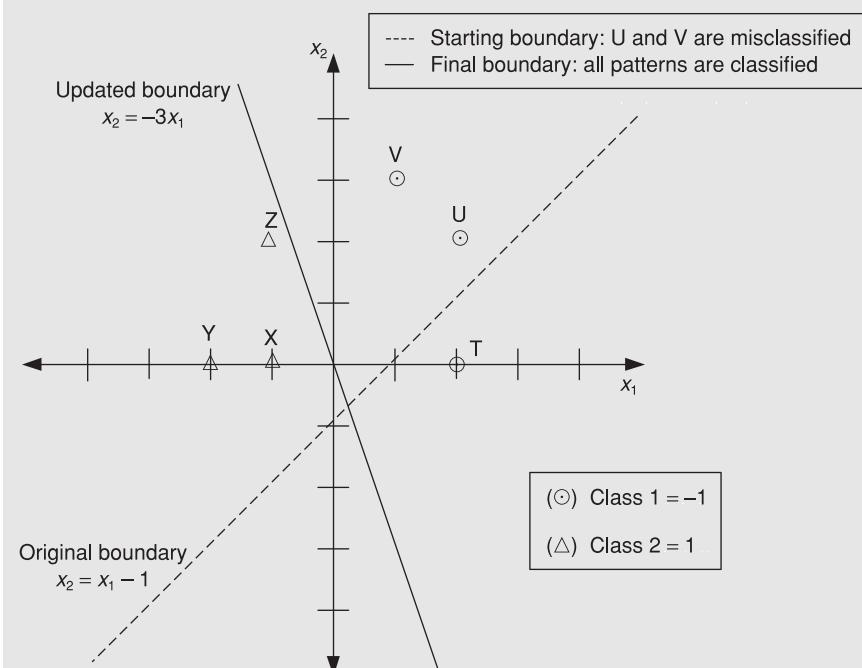
This is another example illustrating the training algorithm for the perceptron and the moving boundary of the classes. Let us presume that  $\eta$  is taken here as 0.5 and there exist two sets of patterns to be classified:

class (1) with target value -1:  $T = [2 \ 0]^\top$ ,  $U = [2 \ 2]^\top$ ,  $V = [1 \ 3]^\top$

class (2) with target value 1:  $X = [-1 \ 0]^\top$ ,  $Y = [-2 \ 0]^\top$ ,  $Z = [-1 \ 2]^\top$

Let us presume that the initial value for  $\theta$  is -1 and the values for  $w_1$  and  $w_2$  are -1 and +1 respectively. This means that the boundary line  $x_2 = -\frac{w_1}{w_2}x_1 + \frac{\theta}{w_2}$

is given initially as  $x_2 = x_1 - 1$  (dashed line in Figure 4.14). Now using the terms of the update weight rule in step 4, we notice that the pattern  $T$  is correctly classified, since  $\text{sgn}(2(-1) + 0 + 1)$  is  $-1$ , which is indeed class (1). So no update is necessary for this exemplar. For the pattern  $U$ ,  $\text{sgn}(2(-1) + 2(1) + 1)$  is equal 1 and the boundary didn't provide for the correct classification, so a weight update is warranted:  $\Delta w_1 = \Delta w_2 = -1(2) = -2$ . The bias update provides for  $\Delta\theta = +1$ . The new values for  $w_1$ ,  $w_2$ , and  $\theta$  become now  $-3$ ,  $-1$ , and  $0$ , respectively. The new boundary is now represented by the line  $x_2 = -3x_1$  (continuous line in Figure 4.14). A brief checking of the remaining patterns shows that no weight and bias updating is required since all of the patterns are now adequately classified.



**Figure 4.14:** Example illustrating the change of the boundary separating classes A and B

Despite some of their learning capabilities, single layer perceptrons were subjected to two major criticisms. They were to hinder the connectionist research in the late 1960s and 1970s. One of them originated from the fact that perceptrons lacked the ability to solve problems dealing with nonlinearly separable patterns. The other criticism was that the perceptron, once trained with a set of training data, cannot adapt its connection weights to a new set of data (lack of generalization). These two main shortcomings of the percep-

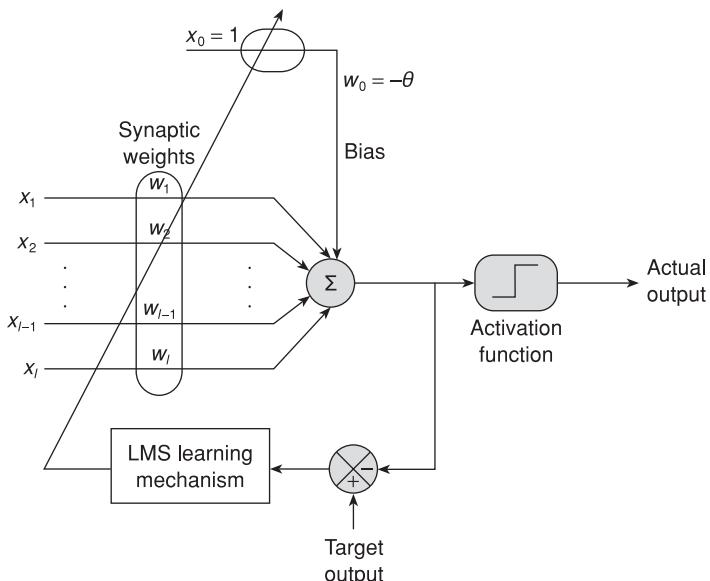
tron were well articulated in Minsky's and Papert's book *Perceptron* [8], in which they had shown that the perceptron did not pass a benchmark test which consists of successfully training a network to solve the logic exclusive (XOR) problem. The publication of *Perceptron* did in fact put on hold the connectionist research for a number of years.

#### 4.4.3 Adaline

Another neural model, more versatile in terms of generalization and more powerful in terms of weight adaptation, was proposed around the same time the perceptron was developed. This connectionist model, also known as the Adaptive Linear Neuron (ADALINE), was developed by Widrow [9] in 1962 and is composed of a linear combiner, an activation function (hard limiter) and a set of trainable signed weights as shown in Figure 4.15. The weights in this model are adjusted according to the least mean square (LMS) algorithm also known as the *Widrow–Hoff learning rule* [5].

While the learning rule for the perceptron is deduced from a quasi-empirical process to obtain an analytical solution for a given decision problem, the learning rule for the adaline is formally derived using the gradient descent algorithm. The LMS rule adjusts the weights by incrementing them every iteration step by an amount proportional to the gradient of the cumulative error of the network  $E(\mathbf{w})$ :

$$\Delta \mathbf{w} = -\eta \nabla_{\mathbf{w}} E(\mathbf{w}) \quad (4.8)$$



**Figure 4.15:** Schematic representation of the adaline

with  $E(\mathbf{w}) = \sum_l \left( \left( t^{(k)} - \left( \sum_i w_i x_i^{(k)} - \theta \right) \right) \right)^2$ . This is the cumulative error for all patterns  $k$ 's ( $k = 1 \dots n$ ) between the desired response  $t^k$  and the actual output  $\left( \sum_i w_i x_i^{(k)} - \theta \right)$  of the linear combiner. As such, the weights are updated individually according to the formulae:

$$\Delta w_i = \eta \left( t^{(k)} - \sum_i w_i x_i^{(k)} \right) x_i^{(k)} \quad (4.9)$$

The steps involving the training of the adaline are quite similar to those used for the perceptron with the difference that the weight updating law (4.9) is being carried out using the desired output and the actual output of the linear combiner before going through the activation function. As such, the training steps of the adaline are given as follows:

*Step 1:* Initialize weights and thresholds to small random values.

*Step 2:* Choose an input–output pattern from the training input–output data set  $(x^{(k)}, t^{(k)})$ .

*Step 3:* Compute the output before activation  $r = \left( \sum_{i=1}^l w_i x_i - \theta \right)$ .

*Step 4:* Adjust the weights according to LMS rule as:

$$\Delta w_i = \eta \left[ t - \left( \sum_i w_i x_i - \theta \right) \right] x_i$$

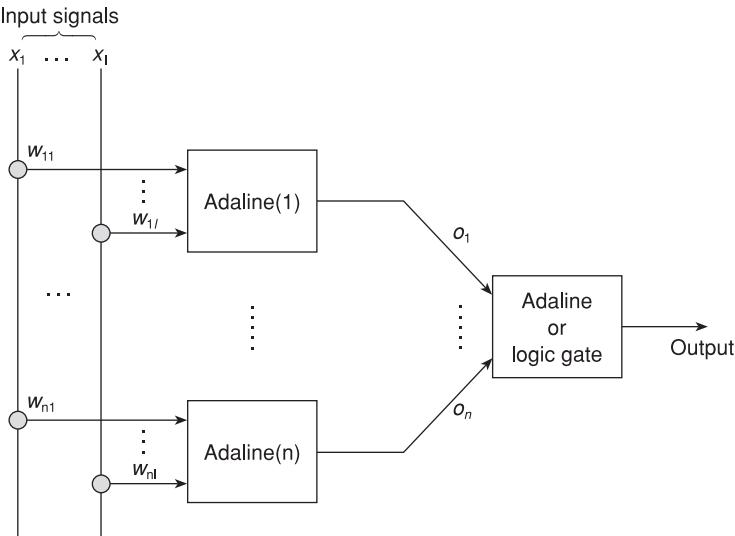
with  $\eta$  being a positive number ranging from 0 to 1 and representing the learning rate.

*Step 5:* Repeat by going to step 2 if the weights do not reach steady-state values.

One very attractive feature of the LMS algorithm is its ease of implementation and especially its ability for generalization, a major feature missing in the perceptron. This means that once an adaline has been trained for a set of patterns, it can be applied successfully to classify a noisy version of the same set.

#### 4.4.4 Madaline

The adaline, while having attractive training capabilities, suffers also (similarly to the perceptron) from the inability to train patterns belonging to non-linearly separable spaces. Researchers have tried to circumvent this difficulty



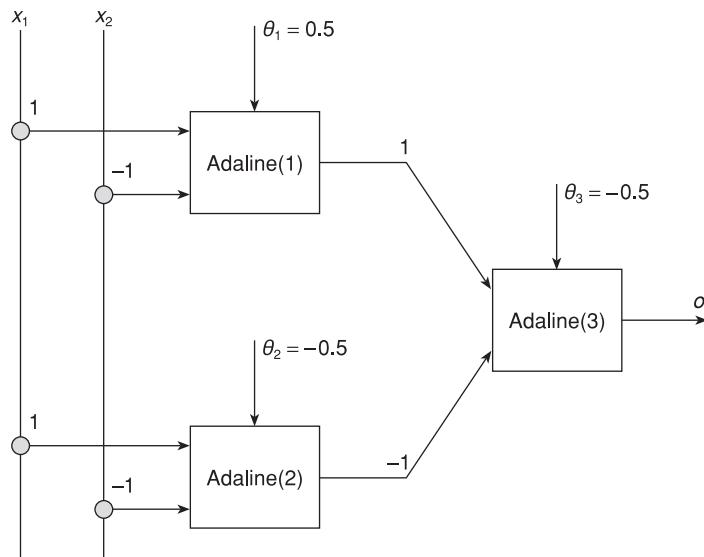
**Figure 4.16:** Schematics of the madaline

by setting cascade layers of adaline units. When first proposed, this seemingly attractive idea didn't lead to much improvement due to the lack of an existing learning algorithm (at the time) capable of adequately updating the synaptic weights of a cascade architecture of perceptrons. Other researchers were able to solve the nonlinear separability problem by combining in parallel a number of adaline units called a *madaline* (Figure 4.16).

For instance, it was shown in [10] that the XOR logic function could be effectively trained by combining in parallel two adaline units using the AND logic gate. The structure shown in Figure 4.17 also executes the XOR logic gate, for which the binary table is given below:

$x_1$	$x_2$	$0 = x_1 \text{XOR} x_2$
0	0	1
0	1	-1
1	0	-1
1	1	1

Despite the successful implementation of the adaline and the madaline units in a number of important applications such as adaptive signal processing and adaptive inverse control [11], many researchers conjectured that if the connectionist computational tools thus developed were to succeed, they would have to be based on neural models involving a topology with a number of cascaded layers. This was later achieved to a large extent with the



**Figure 4.17:** Example of madaline for execution of the “XOR” logic gate

implementation of the backpropagation learning algorithm [12] to neural network models composed of multiple layers of perceptrons. This will be discussed in more detail in the next chapter.

## 4.5 Summary

In this chapter, we presented fundamental concepts for a number of early connectionist models along with brief descriptions of their main features in terms of topology, learning algorithms, and activation functions. An overview of the earliest attempts to build computational neurons was provided along with a historic perspective of what has been achieved in the area since the early 1960s. The pioneering work of McCulloch–Pitts, Rosenblatt, and Widrow paved the way for designing what is recognized today as one of the most active research topics in the area of learning and connectionist modeling. In the following chapters, we provide detailed descriptions of a number of the very often used artificial neural network models along with highlights on their fields of application.

### Problems

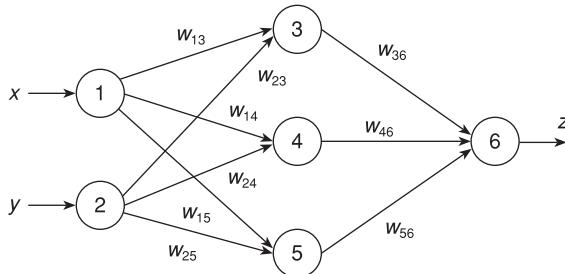
1. Enumerate the different learning algorithms outlined in this chapter and come up with numerical examples to illustrate them.
2. Repeat Example 4.1 by starting with the following initial values  $\theta = -1$ ,  $w_1 = +1$  and  $w_2 = -1$ . In how many stages does the system converge (all patterns are classified)?

3. A perceptron is needed to classify two sets of patterns given as:

class (1):  $T = [3, 0]', U = [2, 1.5]', V = [1, 3]',$  target value  $-1$

class (2):  $X = [-1, 0]', Y = [-2, 0]', Z = [-1, 2]',$  target value  $1$

- Choose  $\eta$  as  $0.5$  and get the boundary line that separates the two classes.
  - Show that the algorithm doesn't converge once the following pattern is added to the system  $W = [-2.5 \ 0]'$ . Explain.
4. The given three-layer network divides the plane with three lines forming a triangle. Calculate the weights that will give a triangle having vertices at  $(x, y)$ , and coordinates  $(0, 0), (1, 3)$ , and  $(3, 1)$ .



5. Discuss the major objections given by Minsky and Papert in their textbook *Perceptron* over the structure of the perceptron. Was the claim of the authors that “any structure with any number of perceptrons is not able to solve the problem of classification of nonlinear separable systems” justified? Discuss this.
6. In the adaline training, instead of using the output of the combiner in the cumulative error, one may also pass this output through a smooth function. Derive the learning rule for an adaline for which the combiner output goes through the sigmoid function given by:
- $$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$
7. Repeat problem 6 by using  $\tanh(x)$  as the activation function.
8. Give a structure of the madaline that is able to solve the “XOR” problem.  
Hint: One may use two parallel adalines connected with an “AND” gate at their output.
9. The normalized Widrow–Hoff learning rule, also known as the normalized least mean square learning rule, is expressed as:

$$\Delta \mathbf{w}^{(k)} = \eta(t^{(k)} - \mathbf{w}^{(k)} \mathbf{x}^{(k)}) \frac{\mathbf{x}^{(k)}}{\|\mathbf{x}^{(k)}\|^2}$$

where  $\Delta \mathbf{w}^{(k)} = \mathbf{w}^{(k+1)} - \mathbf{w}^{(k)}$  represents the weight vector change from iteration  $(k)$  to iteration  $(k+1)$ ,  $\|\mathbf{x}^{(k)}\|$  is the Euclidean norm of the input vector  $\mathbf{x}^{(k)}$  at iteration  $(k)$ ,  $t^{(k)}$  is the target data at iteration  $(k)$  and  $\eta$  is a positive number ranging from  $0$  to  $1$ :  $0 < \eta < 1$ .

Show that if the same input vector  $\mathbf{x}^{(k)}$  is presented at iteration  $(k+1)$ , then the weight vector decreases by the factor  $(1 - \eta)$  going from iteration  $(k)$  to iteration  $(k+1)$ . That is  $\Delta \mathbf{w}^{(k+1)} = (1 - \eta) \mathbf{w}^{(k)}$ .

## References

1. Hopgood, A. (1993) *Knowledge-based Systems for Engineers and Scientists*, Boca Raton, Florida, CRC Press, 159–85.
2. Jang, J., Sun, S., and Mizutani, E. (1997) *Neuro-Fuzzy and Soft Computing*, Prentice Hall, Upper Saddle River, NJ.
3. Haykin, S. (1994) *Neural Networks, A Comprehensive Foundation*, MacMillan Publishing, Englewood Cliffs, NJ.
4. Werbos, P. (1974) “Beyond regression: new tools for prediction and analysis in the behavioral sciences,” PhD dissertation, Harvard University.
5. Widrow, B., and Hoff Jr., M. (1960) “Adaptive switching circuits,” *IRE WESCON Convention Record*, pp. 96–104.
6. Pitts, W., and McCulloch, W.S. (1947) “How we know universals: the perception of auditory and visual forms,” *Bulletin Math. Biophys.*, 9: 27–47.
7. Rosenblatt, F. (1962) *Principles of Neurodynamics*, Spartan Books, Washington, D.C.
8. Minsky, M.L., and Papert, S. (1969) *Perceptrons: An Introduction to Computational Geometry*, MIT Press, Cambridge, MA, USA.
9. Widrow, B. (1959) “Generalization and information storage in networks of adaline ‘neurons’,” in *Self Organizing Systems*, M.C. Yovitz, G.T. Jacobi, and G.D. Goldstein (eds), pp. 435–61, Spartan Books, Washington, D.C.
10. Widrow, B., and Winter, R. (1988) “Neural nets for adaptive filtering and adaptive pattern recognition,” *IEEE Computer*, no. 21, vol. 3, pp. 25–39.
11. Widrow, B., and Stearns, D. (1985) *Adaptive Signal Processing*, Prentice Hall, Upper Saddle River, NJ.
12. Rumelhart, D., Hinton, G., and Williams, R. (1986) “Learning representations by backpropagation errors,” *Nature*, vol. 323, pp. 533–6.

# Major classes of neural networks

## Chapter outline

5.1	Introduction	249
5.2	The multilayer perceptron	250
5.3	Radial basis function networks	263
5.4	Kohonen's self-organizing network	268
5.5	The Hopfield network	274
5.6	Industrial and commercial applications of ANN	281
5.7	Summary	289
	Problems	290
	References	293

## 5.1 Introduction

As mentioned previously, the single layer *perceptron* designed by Rosenblatt has a major deficiency because it lacks the important capability of recognizing patterns belonging to non-separable linear spaces. This severe restriction of the perceptron and the lack of powerful learning algorithms dealing with more complex connectionist structures effectively led to the diminishing interest in connectionist research in the late 1960s and most of the 1970s. The *madaline*, while quite powerful in terms of adaptive learning capabilities and reasonably more versatile in solving a wider range of problems than the single layer perceptron, was still restricted in its capabilities for dealing with complex functional mappings and multi-class pattern recognition problems. In the mid-1980s, the reformulation of the backpropagation learning algorithm, originally developed by Werbos in 1974 [1], along with its implementation as a multilayer neural structure, led to breakthroughs in dealing with problems involving non-separable linear spaces. This ultimately led to considerable renewed interest in the field of neural networks. Many other topologies and learning algorithms have since been developed and have tackled with success

an ever-increasing number of complex problems never thought to be completely solved in the early days of neural networks. In this chapter, we outline the main features of a number of popular neural networks and provide an overview on their topologies and their learning capabilities. These include the multilayer perceptron, the radial basis function network, the Kohonen self-organizing map, and the Hopfield network. At the end of the chapter a set of industrial applications using neural networks are outlined, ranging from process control to communication systems. A good number of references is also included for guiding the reader on the appropriate system to choose when tackling a particular application within a given field.

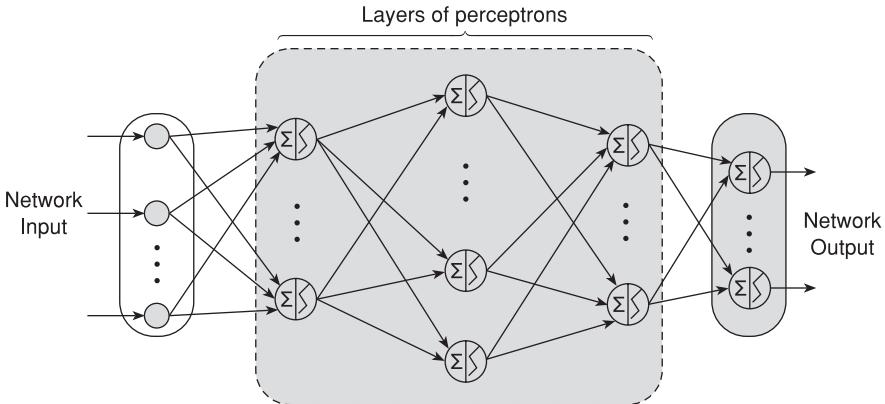
## 5.2 The multilayer perceptron

### 5.2.1 Topology

The multilayer perceptron (MLP) belongs to the class of feedforward networks, meaning that the information flows among the network nodes exclusively in the forward direction. Figure 5.1 illustrates a typical representation of a multilayer perceptron with an input layer, three hidden layers, and an output layer. This structure was first proposed in the 1960s to circumvent the separability problem of the earlier models of the perceptron and the adaline. However, the lack of efficient learning algorithms to tackle such a topology hampered the development of the multilayer model during that time. The number of hidden layers required within a multilayer perceptron depends in major part on the type of problem being addressed. In fact, a formal explicit theory, which states how many hidden layers are needed in a given network to solve a specified task, does not fully exist yet. However, the larger the number, the more classes the system can handle, although this increase in layers could come at the expense of some other issues mainly related to convergence behavior of the learning algorithm. For instance, a system with a single hidden layer is able to solve the problem of the XOR function or other related problems in which the separation boundaries are relatively simple. It was also shown in the work of Cybenko [2] that an MLP network with one single hidden layer composed of an *appropriate* number of nodes with sigmoid activation functions can approximate any type of continuous mapping on a compact set.

### 5.2.2 Backpropagation learning algorithm

It was only in the mid-1980s that the multilayer architecture, first proposed in the late 1960s, re-emerged as a solid connectionist model to solve a wide range of complex problems. This occurred following the reformulation of a powerful learning algorithm commonly called *backpropagation learning* (BPL), which was originally developed by Werbos in 1974 [1]. It was later implemented in the multilayer perceptron topology with a great deal of success [3]. The algorithm is based on the gradient descent technique for solving



**Figure 5.1:** Schematic representation of the MLP network

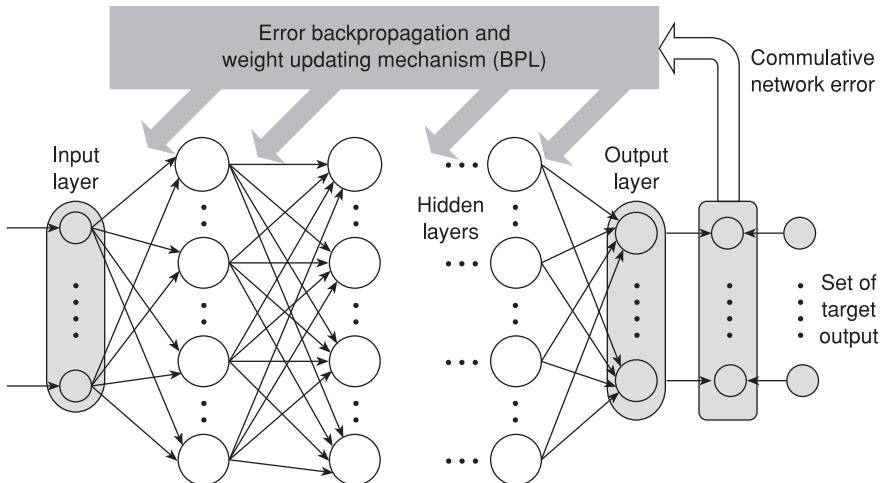
an optimization problem, which involves the minimization of the network cumulative error  $E_c$ .  $E_c$  represents the sum of  $n$  squared errors (Euclidian norm)  $E(k)$ 's with  $E(k) = \frac{1}{2} \sum_{i=1}^q [t_i(k) - o_i(k)]^2$  being the square of the Euclidian

norm of the vectorial difference between the  $k$ -th target output vector  $t(k)$  and the  $k$ -th actual output vector  $o(k)$  of the network.  $n$  is the number of training patterns presented to the network for learning purposes. The algorithm is designed in such a way as to update the weights in the direction of the gradient descent of the cumulative error (with respect to the weight vector). This is done in an iterative way. At the start, patterns are presented to the network. A feedback signal (error between the target signals and the actual output) is then propagated backward with the main task of updating the weights of the layers' connections according to a well-defined algorithm commonly known as the backpropagation learning algorithm. Figure 5.2 illustrates the mechanism of the error backpropagation.

Formally, the backpropagation algorithm is carried out as follows. Using a similar notation to that of the previous chapter, and using the sigmoid function as the activation function for all the neurons of the network, we define  $E_c$  as

$$E_c = \sum_{k=1}^n E(k) = \frac{1}{2} \sum_{k=1}^n \sum_{i=1}^q [t_i(k) - o_i(k)]^2 \quad (5.1)$$

where the index  $i$  represents the  $i$ -th neuron of the output layer composed of a total number of  $q$  neurons. The formulation of the optimization problem can now be stated as finding the set of network weights that minimizes  $E_c$  or  $E(k)$ . This depends on whether the learning is made off line (all the training patterns are presented to the system at once) or on line (training is made pattern by pattern). For the first case the optimization problem is stated as:



**Figure 5.2:** Schematic representation of the MLP network illustrating the notion of error backpropagation

$$\min_{\mathbf{w}} E_c = \min_{\mathbf{w}} \frac{1}{2} \sum_{k=1}^n \sum_{i=1}^q [t_i(k) - o_i(k)]^2 \quad (5.2)$$

while for the second case, the optimization problem is formulated as:

$$\min_{\mathbf{w}} E(k) = \min_{\mathbf{w}} \frac{1}{2} \sum_{i=1}^q [t_i(k) - o_i(k)]^2 \quad (5.3)$$

In both formulations, the vector  $\mathbf{w}$  denotes the network weight vector with components corresponding to the interconnection weights among all the neurons of the network. Several authors have discussed and derived the BPL algorithm in ample detail. The reader may wish to consult for this purpose reference [4]. We derive here the updating rules allowing for the minimization of the cost functions in (5.2) or (5.3). For the case of on line training, the generalized delta rule becomes expressed as

$$\Delta \mathbf{w}^{(l)} = -\eta \frac{\partial E(k)}{\partial \mathbf{w}^{(l)}} \quad (5.4)$$

where  $\frac{\partial E(k)}{\partial \mathbf{w}^{(l)}}$  is the gradient of the error  $E(k)$  with respect to the vector  $\mathbf{w}^{(l)}$  corresponding to all interconnection weights between layer  $(l)$  and the preceding layer  $(l-1)$ .  $\eta$  denotes the learning rate parameter which is a small positive number (usually between 0 and 1) reflecting the convergence and stability behaviors of the learning algorithm and  $\Delta \mathbf{w}^{(l)}$  denotes the difference between the vectors  $\mathbf{w}^{(l)}(k+1)$  and  $\mathbf{w}^{(l)}(k)$  representing the interconnection weights leading to neurons at layer  $(l)$  after and before the presentation of the

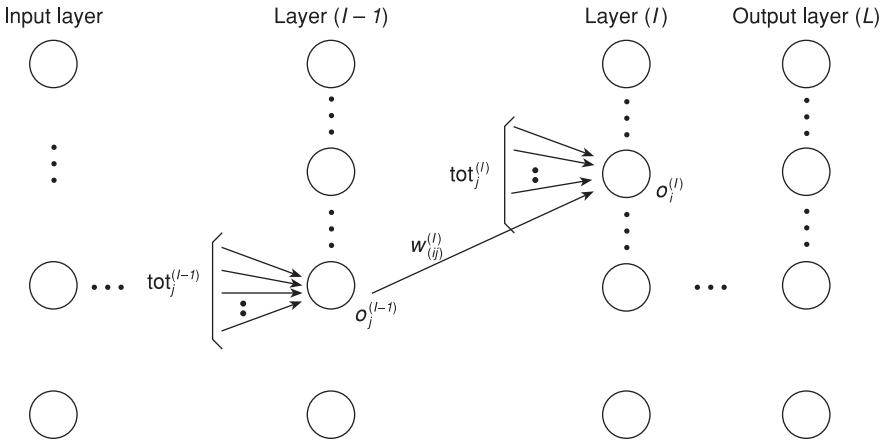


Figure 5.3: Illustration of interconnection between layers of MLP

training pattern  $k$ , respectively. Specifically,  $\Delta w_{ij}^{(l)}$  is the weight update for the connection linking node  $j$  of layer  $(l - 1)$  to node  $i$  located at layer  $l$  and  $o_j^{(l-1)}$  represents the output of the  $j$ -th neuron at layer  $(l - 1)$  (the one located just before layer  $l$ ). The signal  $\text{tot}_i^{(l)}$  represents the sum of all signals reaching node  $(i)$  at hidden layer  $(l)$  coming from previous layer  $(l - 1)$ . All notations used are summarized in Figure 5.3.

Using chain rule differentiation we obtain:

$$\Delta \mathbf{w}^{(l)} = \Delta w_{ij}^{(l)} = -\eta \left[ \frac{\partial E(k)}{\partial o_i^{(l)}} \right] \left[ \frac{\partial o_i^{(l)}}{\partial \text{tot}_i^{(l)}} \right] \left[ \frac{\partial \text{tot}_i^{(l)}}{\partial w_{ij}^{(l)}} \right] \quad (5.5)$$

For the case where layer  $(l)$  is the output layer  $(L)$ , equation (5.5) becomes expressed as:

$$\Delta w_{ij}^{(L)} = \eta [t_i - o_i^{(L)}] [f'(\text{tot}_i^{(L)})] o_j^{(L-1)} \quad (5.6)$$

$$\text{where } f'(\text{tot}_i^{(l)}) = \frac{\partial f(\text{tot}_i^{(l)})}{\partial \text{tot}_i^{(l)}}$$

If we denote  $\delta_i^{(L)} = [t_i - o_i^{(L)}] [f'(\text{tot}_i^{(L)})]$  as being the error signal of the  $i$ -th node of the output layer, we end up with the expression of the weight update at layer  $(L)$ :

$$\Delta w_{ij}^{(L)} = \eta \delta_i^{(L)} o_j^{(L-1)} \quad (5.7)$$

In the case where  $f$  is the sigmoid function, the error signal becomes expressed as:

$$\delta_i^{(L)} = [(t_i - o_i^L) o_i^L (1 - o_i^L)] \quad (5.8)$$

Propagating the error backward now, and for the case where  $(l)$  represents a hidden layer ( $l < L$ ), the expression of  $\Delta w_{ij}^{(l)}$  becomes given by:

$$\Delta w_{ij}^{(l)} = \eta \delta_i^{(l)} o_j^{(l-1)} \quad (5.9)$$

where the error signal  $\delta_i^{(l)}$  is now expressed as a function of output of previous layers as:

$$\delta_i^{(l)} = f'(\text{tot}_i^{(l)}) \sum_{p=1}^{n_l} \delta_p^{(l+1)} w_{p_i}^{(l+1)} \quad (5.10)$$

Again when  $f$  is taken as the sigmoid function,  $\delta_i^{(l)}$  becomes expressed as:

$$\delta_i^{(l)} = o_i^{(l)} (1 - o_i^{(l)}) \sum_{p=1}^{n_l} \delta_p^{(l+1)} w_{p_i}^{(l+1)} \quad (5.11)$$

The index  $n_l$  in equations (5.10) and (5.11) denotes the total number of neurons in layer  $(l + 1)$ , the index  $i$  represents neurons of layer  $(l)$  and the index  $p$  represents the neurons of layer  $(l + 1)$ . The fact that  $\delta_i^{(l)}$  in (5.10) gets its value from knowledge of  $\delta_p^{(l+1)}$  of the next layer gives the algorithm its name as the backpropagation learning algorithm (BPL).

For the case of off-line training, the weight update rule becomes expressed as:

$$\Delta \mathbf{w}^{(l)} = -\eta \frac{\partial E_c}{\partial \mathbf{w}^{(l)}} \quad (5.12)$$

All previous steps outlined for developing the on line update rules are reproduced here with the exception that  $E(k)$  becomes replaced with  $E_c$ . In both cases, though, once the network weights have reached steady state values, the training algorithm is said to converge. At every training cycle, and as can be seen from equations 5.6 and 5.7, the values of the weights are propagated backward, starting from the output layer, all the way to the first layer of the network structure. While a larger value of  $\eta$  may speed up the convergence of the algorithm, it has been observed that this may lead to oscillations. Several techniques have been proposed in the literature to speed up the BPL while overcoming the convergence stability problem. Some of the techniques proposed involve the addition of the so-called momentum term involving the weight vector at iteration  $k - 1$  [5], while others involve normalization procedures for the update formulae outlined earlier. Other speed enhancement approaches have also been proposed in [6]. To summarize, we provide here the steps involved in training an MLP network using the BPL algorithm:

*Step 1:* Initialize weights and thresholds to small random values.

*Step 2:* Choose an input–output pattern from the training input–output data set  $(\mathbf{x}(k), \mathbf{t}(k))$ .

*Step 3: Propagate the  $k$ -th signal forward through the network and compute the output values for all  $i$  neurons at every layer ( $l$ ) using  $o_i^l(k) = f\left(\sum_{p=0}^{n_l-1} w_{ip}^l o_p^{l-1}\right)$ .*

**Step 4:** Compute the total error value  $E = E(k) + E$  and the error signal  $\delta_i^{(L)}$  using the formulae  $\delta_i^{(L)} = [t_i - o_i^L][f'(\text{tot}_i^{(L)})]$ .

*Step 5:* Update the weights according to  $\Delta w_{ij}^{(l)} = -\eta \delta_i^{(l)} o_j^{(l-1)}$  for  $l = L, \dots, 1$  using  $\delta_i^{(L)} = [t_i - o_i^L][f'(\text{tot}_i^{(L)})]$  and proceeding backward using  $\delta_i^{(l)} = o_i^{(l)}(1 - o_i^{(l)}) \sum_{p=1}^{n_l} \delta_p^{(l+1)} w_{pi}^{(l+1)}$  for  $l < L$ .

*Step 6:* Repeat the process starting from step 2 using another exemplar. Once all exemplars have been used, we then reach what is known as one-epoch training.

*Step 7:* Check if the cumulative error  $E$  in the output layer has become less than a predetermined value. If so we conclude the network has been trained. If not, repeat the whole process for one more epoch.

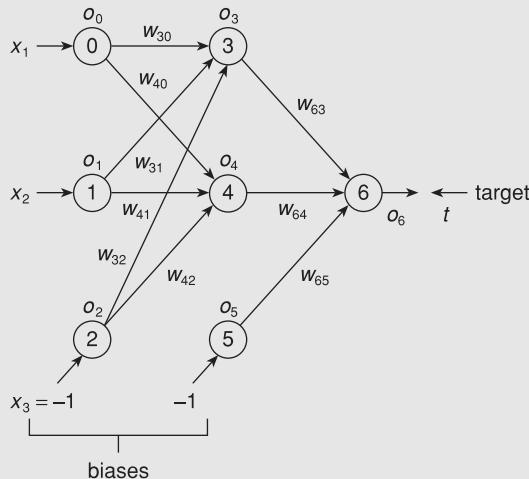
### Example 5.1

To illustrate this powerful algorithm, we apply it for the training of the following network shown in Figure 5.4. The following three training pattern pairs are used, with  $\mathbf{x}$  and  $\mathbf{t}$  being the input and the output data respectively:

$$\mathbf{x}^{(1)} = (0.3, 0.4), \quad \mathbf{t}^{(1)} = (0.88),$$

$$\mathbf{x}^{(2)} = (0.1, 0.6), \quad \mathbf{t}^{(2)} = (0.82),$$

$$\mathbf{x}^{(3)} = (0.9, 0.4), \quad \mathbf{t}^{(3)} = (0.57),$$



**Figure 5.4:** Structure of the neural network of Example 5.1

Biases are treated here as connection weights that are always multiplied by  $-1$  through a neuron to avoid special case calculation for biases. Each neuron uses a unipolar sigmoid activation function given by:

$$o = f(\text{tot}) = \frac{1}{1 + e^{-\lambda \text{tot}}}, \text{ using } \lambda = 1, \text{ then } f'(\text{tot}) = o(1 - o)$$

### Step (1) – Initialization

- Initialize the weights to small random values. We assume all weights are initialized to 0.2; set learning rate to  $\eta = 0.2$ ; set maximum tolerable error to  $E_{\max} = 0.01$  (i.e., 1% error); set current error value to  $E = 0$ ; set current training pattern to  $k = 1$ .

### *Training Loop – Loop (1)*

#### Step (2) – Apply input pattern

- Apply the 1st input pattern to the input layer:

$$\mathbf{x}^{(1)} = (0.3, 0.4), \mathbf{t}^{(1)} = (0.88), \text{ then, } o_0 = \mathbf{x}_1 = 0.3; o_1 = \mathbf{x}_2 = 0.4; o_2 = \mathbf{x}_3 = -1$$

#### Step (3) – Forward propagation

- Propagate the signal forward through the network:

$$o_3 = f(w_{30}o_0 + w_{31}o_1 + w_{32}o_2) = 0.4850$$

$$o_4 = f(w_{40}o_0 + w_{41}o_1 + w_{42}o_2) = 0.4850$$

$$o_5 = -1$$

$$o_6 = f(w_{63}o_3 + w_{64}o_4 + w_{65}o_5) = 0.4985$$

#### Step (4) – Output error measure

- Compute the error value  $E$  and the error signal  $\delta_6$  of the output layer:

$$E = \frac{1}{2}(t - o_6)^2 + E = 0.0728$$

$$\delta_6 = f'(\text{tot}_6)(t - o_6)$$

$$= o_6(1 - o_6)(t - o_6)$$

$$= 0.0954$$

#### Step (5) – Error backpropagation

- Propagate the errors backward to update the weights and compute the error signals of the preceding layers.

Third layer weight updates:

$$\Delta w_{63} = \eta \delta_6 o_3 = 0.0093$$

$$w_{63}^{\text{new}} = w_{63}^{\text{old}} + \Delta w_{63} = 0.2093$$

$$\Delta w_{64} = \eta \delta_6 o_4 = 0.0093$$

$$w_{64}^{\text{new}} = w_{64}^{\text{old}} + \Delta w_{64} = 0.2093$$

$$\Delta w_{65} = \eta \delta_6 o_5 = -0.0191$$

$$w_{65}^{\text{new}} = w_{65}^{\text{old}} + \Delta w_{65} = 0.1809$$

Second layer error signals:

$$\delta_3 = f'_3(\text{tot}_3) \sum_{i=6}^6 w_{i3} \delta_i = o_3(1 - o_3) w_{63} \delta_6 = 0.0048$$

$$\delta_4 = f'_4(\text{tot}_4) \sum_{i=6}^6 w_{i4} \delta_i = o_4(1 - o_4) w_{64} \delta_6 = 0.0048$$

Second layer weight updates:

$$\Delta w_{30} = \eta \delta_3 o_0 = 0.00028586$$

$$w_{30}^{\text{new}} = w_{30}^{\text{old}} + \Delta w_{30} = 0.2003$$

$$\Delta w_{31} = \eta \delta_3 o_1 = 0.00038115$$

$$w_{31}^{\text{new}} = w_{31}^{\text{old}} + \Delta w_{31} = 0.2004$$

$$\Delta w_{32} = \eta \delta_3 o_2 = -0.00095288$$

$$w_{32}^{\text{new}} = w_{32}^{\text{old}} + \Delta w_{32} = 0.1990$$

$$\Delta w_{40} = \eta \delta_4 o_0 = 0.00028586$$

$$w_{40}^{\text{new}} = w_{40}^{\text{old}} + \Delta w_{40} = 0.2003$$

$$\Delta w_{41} = \eta \delta_4 o_1 = 0.00038115$$

$$w_{41}^{\text{new}} = w_{41}^{\text{old}} + \Delta w_{41} = 0.2004$$

$$\Delta w_{42} = \eta \delta_4 o_2 = -0.00095288$$

$$w_{42}^{\text{new}} = w_{42}^{\text{old}} + \Delta w_{42} = 0.1990$$

### **Training Loop – Loop (2)**

#### **Step (2) – Apply the 2nd input pattern**

Apply the 2nd input pattern to the input layer:

$$\mathbf{x}^{(2)} = (0.1, 0.6), \mathbf{t}^{(2)} = (0.82), \text{then, } o_0 = 0.1, o_1 = 0.6, o_2 = -1$$

#### **Step (3) – Forward propagation**

$$o_3 = f(w_{30}o_0 + w_{31}o_1 + w_{32}o_2) = 0.4853$$

$$o_4 = f(w_{40}o_0 + w_{41}o_1 + w_{42}o_2) = 0.4853$$

$$o_5 = -1$$

$$o_6 = f(w_{63}o_3 + w_{64}o_4 + w_{65}o_5) = 0.5055$$

#### **Step (4) – Output error measure**

$$E = \frac{1}{2}(t - o_6)^2 = 0.1222$$

$$\delta_6 = o_6(1 - o_6)(t - o_6) = 0.0786$$

**Step (5) – Error backpropagation**

Third layer weight updates:

$$\Delta w_{63} = \eta \delta_6 o_3 = 0.0076$$

$$w_{63}^{\text{new}} = w_{63}^{\text{old}} + \Delta w_{63} = 0.2169$$

$$\Delta w_{64} = \eta \delta_6 o_4 = 0.0076$$

$$w_{64}^{\text{new}} = w_{64}^{\text{old}} + \Delta w_{64} = 0.2169$$

$$\Delta w_{65} = \eta \delta_6 o_5 = -0.0157$$

$$w_{65}^{\text{new}} = w_{65}^{\text{old}} + \Delta w_{65} = 0.1652$$

Second layer error signals:

$$\delta_3 = f'_3(\text{tot}_3) \sum_{i=6}^6 w_{i3} \delta_i = o_3(1 - o_3) w_{63} \delta_6 = 0.0041$$

$$\delta_4 = f'_4(\text{tot}_4) \sum_{i=6}^6 w_{i4} \delta_i = o_4(1 - o_4) w_{64} \delta_6 = 0.0041$$

Second layer weight updates:

$$\Delta w_{30} = \eta \delta_3 o_0 = 0.000082169$$

$$w_{30}^{\text{new}} = w_{30}^{\text{old}} + \Delta w_{30} = 0.2004$$

$$\Delta w_{31} = \eta \delta_3 o_1 = 0.00049302$$

$$w_{31}^{\text{new}} = w_{31}^{\text{old}} + \Delta w_{31} = 0.2009$$

$$\Delta w_{32} = \eta \delta_3 o_2 = -0.00082169$$

$$w_{32}^{\text{new}} = w_{32}^{\text{old}} + \Delta w_{32} = 0.1982$$

$$\Delta w_{40} = \eta \delta_4 o_0 = 0.000082169$$

$$w_{40}^{\text{new}} = w_{40}^{\text{old}} + \Delta w_{40} = 0.2004$$

$$\Delta w_{41} = \eta \delta_4 o_1 = 0.00049302$$

$$w_{41}^{\text{new}} = w_{41}^{\text{old}} + \Delta w_{41} = 0.2009$$

$$\Delta w_{42} = \eta \delta_4 o_2 = -0.00082169$$

$$w_{42}^{\text{new}} = w_{42}^{\text{old}} + \Delta w_{42} = 0.1982$$

**Training Loop – Loop (3)****Step (2) – Apply the 3rd input pattern to the input layer**

$$\mathbf{x}^{(2)} = (0.9, 0.4), \mathbf{t}^{(2)} = (0.57), \text{then, } o_0 = 0.9, o_1 = 0.4, o_2 = -1$$

**Step (3) – Forward propagation**

$$o_3 = f(w_{30} o_0 + w_{31} o_1 + w_{32} o_2) = 0.5156$$

$$o_4 = f(w_{40} o_0 + w_{41} o_1 + w_{42} o_2) = 0.5156$$

$$o_5 = -1$$

$$o_6 = f(w_{63} o_3 + w_{64} o_4 + w_{65} o_5) = 0.5146$$

**Step (4) – Output error measure**

$$E = \frac{1}{2}(t - o_6)^2 + E = 0.1237$$

$$\delta_6 = o_6(1 - o_6)(t - o_6) = 0.0138$$

### Step (5) – Error backpropagation

Third layer weight updates:

$$\Delta w_{63} = \eta \delta_6 o_3 = 0.0014$$

$$w_{63}^{\text{new}} = w_{63}^{\text{old}} + \Delta w_{63} = 0.2183$$

$$\Delta w_{64} = \eta \delta_6 o_4 = 0.0014$$

$$w_{64}^{\text{new}} = w_{64}^{\text{old}} + \Delta w_{64} = 0.2183$$

$$\Delta w_{65} = \eta \delta_6 o_5 = -0.0028$$

$$w_{65}^{\text{new}} = w_{65}^{\text{old}} + \Delta w_{65} = 0.1624$$

Second layer error signals:

$$\delta_3 = f'_3(\text{tot}_3) \sum_{i=6}^6 w_{i3} \delta_i = o_3(1 - o_3) w_{63} \delta_6 = 0.00074948$$

$$\delta_4 = f'_4(\text{tot}_4) \sum_{i=6}^6 w_{i4} \delta_i = o_4(1 - o_4) w_{64} \delta_6 = 0.00074948$$

Second layer weight updates:

$$\Delta w_{30} = \eta \delta_3 o_0 = 0.00013491$$

$$w_{30}^{\text{new}} = w_{30}^{\text{old}} + \Delta w_{30} = 0.2005$$

$$\Delta w_{31} = \eta \delta_3 o_1 = 0.000059958$$

$$w_{31}^{\text{new}} = w_{31}^{\text{old}} + \Delta w_{31} = 0.2009$$

$$\Delta w_{32} = \eta \delta_3 o_2 = -0.00014990$$

$$w_{32}^{\text{new}} = w_{32}^{\text{old}} + \Delta w_{32} = 0.1981$$

$$\Delta w_{40} = \eta \delta_4 o_0 = 0.00013491$$

$$w_{40}^{\text{new}} = w_{40}^{\text{old}} + \Delta w_{40} = 0.2005$$

$$\Delta w_{41} = \eta \delta_4 o_1 = 0.000059958$$

$$w_{41}^{\text{new}} = w_{41}^{\text{old}} + \Delta w_{41} = 0.2009$$

$$\Delta w_{42} = \eta \delta_4 o_2 = -0.00014990$$

$$w_{42}^{\text{new}} = w_{42}^{\text{old}} + \Delta w_{42} = 0.1981$$

### Step (6) – One-epoch looping

The training patterns have been cycled once, which is called one *epoch*, so we stop the loop and continue with step (6) below.

### Step (7) – Total error checking

Check whether the current total error is acceptable. If  $E < E_{\max}$  then terminate the training process and output the final weights. Otherwise  $E = 0$ ,  $k = 1$ , and initiate the new training epoch by going to Step (1). In this example,  $E = 0.1237$  and  $E_{\max} = 0.01$ , which means that we have to continue with the next epoch by cycling the training data again. We keep the training process going until the desired performance goal is met (i.e.,  $E < E_{\max}$ ) or until a specified number of epochs has been reached.

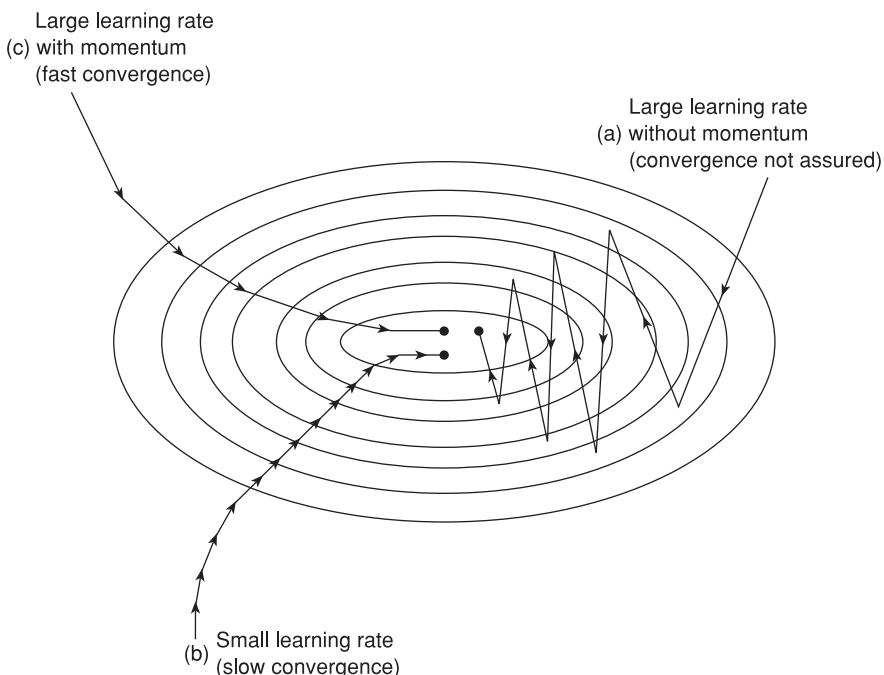
### 5.2.3 Momentum

The gradient descent usually requires infinitesimal differentiation steps. For small values of the learning parameter  $\eta$ , this leads most often to a very slow convergence rate of the algorithm. Larger learning parameters have been known to lead to unwanted oscillations in the weight space and may even cause divergence of the algorithm.

To avoid these issues, researchers have devised a modified weight updating algorithm in which the change of the weight of the upcoming iteration (at time  $t + 1$ ) is made dependent on the weight change of the current iteration (at time  $t$ ). This provides for an updated weight leading to a uniform decrease of the error function instead of keeping oscillating as is the case for large values of the learning parameter. This is done by adding to equation (5.12) a term known as the momentum, which leads to the modified weight update formulae given as

$$\Delta \mathbf{w}^{(l)}(t+1) = -\eta \frac{\partial E_c(t)}{\partial \mathbf{w}^{(l)}} + \gamma \Delta \mathbf{w}^{(l)}(t) \quad (5.13)$$

The contribution of the past weight change is weighed by the factor  $\gamma$  belonging to  $[0, 1]$ , which is usually taken in the range of 0.9 or 0.8. Figure 5.5 illustrates the behavior of the weight change with and without momentum addition.



**Figure 5.5:** Effect of the momentum on the rate of convergence of the weights

### Example 5.2

#### *Effect of hidden nodes on function approximation*

To illustrate the effects of the number of hidden neurons on the approximation capabilities of the MLP, we use here the simple function  $f(x)$  given by:

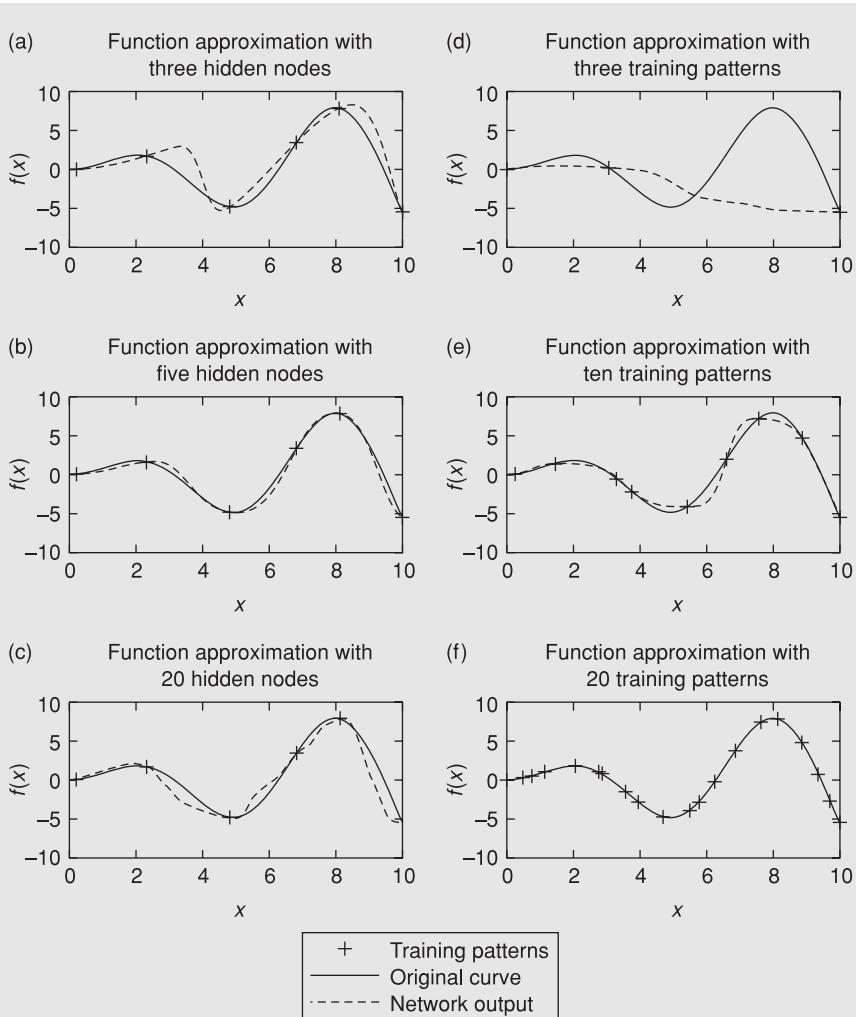
$$f(x) = x \sin(x)$$

Six input/output samples were selected from the range [0:10] of the variable  $x$ . The first run was made for a network with three hidden nodes. The results are shown in Figure 5.6(a). Another run was made for a network with five (Figure 5.6(b)) and 20 (Figure 5.6(c)) nodes respectively. From the result of the simulation, one may conclude that a higher number of nodes is not always better as is seen in Figure 5.6(c). This is mostly due to the fact that a network with this structure has overinterpolated in between the samples and we say that the network was overtrained. This happens when the network starts to memorize the patterns instead of interpolating between them. In this series of simulations the best match with the original curve was obtained with a network having five hidden nodes. It seems here that this network (with five nodes) was able to interpolate quite well the nonlinear behavior of the curve. A smaller number of nodes didn't permit a faithful approximation of the function given that the nonlinearities induced by the network were not enough to allow for adequate interpolation between samples.

### Example 5.3

#### *Effect of training patterns on function approximation*

The same function was approximated with a network with a fixed number of nodes (taken as five here), but with a variable number of training patterns. The first run with three samples (Figure 5.6(d)) was not able to provide a good match with the original curve. This can be explained by the fact that the three patterns, in the case of a nonlinear function such as this, are not able to reproduce the relatively high nonlinearities of the function. A higher number of training patterns provided better results as shown in Figure 5.6(e). The best result was obtained for the case of 20 training patterns as shown in Figure 5.6(f). This is due to the fact that a network with five hidden nodes interpolates extremely well in between close training patterns.



**Figure 5.6 a, b, c:** Effect of the number of hidden nodes on MLP approximation of  $f(x)$

**Figure 5.6 d, e, f:** Effect of the number of training patterns on MLP approximation of  $f(x)$

#### 5.2.4 Applications and limitations of MLPs

Multilayer perceptrons are currently among the most used connectionist models. This stems from the relative ease of training and implementing them, either in hardware or software forms. MLPs have been used in a wide variety of applications such as in signal processing, weather forecasting, financial market prediction, pattern recognition, signal compression, to name a few. This being said, it is well known that MLPs are not a panacea to resolve all

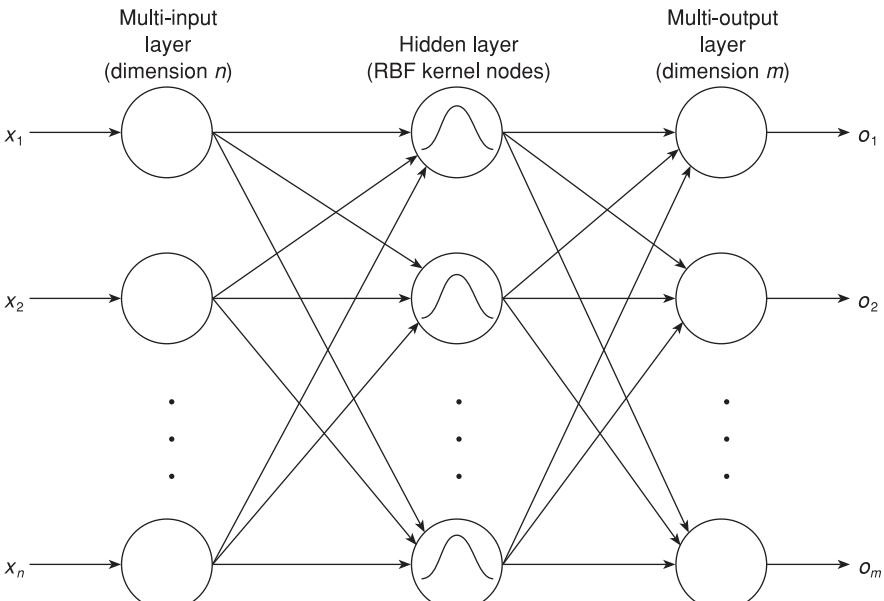
types of problems involving processes with nonlinear behavior or noisy data. Among the well-known problems that may hinder the generalization or approximation capabilities of MLPs is the one related to the convergence behavior of the connection weights during the learning stage. In fact, the gradient-descent-based algorithm used to update the network weights may never converge to the global minima. This is particularly true in the case of highly nonlinear behavior of the system being approximated by the network. Many remedies have been proposed to tackle this issue either by retraining the network a number of times or by using optimization techniques such as those based on genetic algorithms or simulated annealing.

## 5.3 Radial basis function networks

### 5.3.1 Topology

Radial basis function networks (RBFN) represent a special category of the feedforward neural networks architecture. Inspired by the powerful functions of the biological receptive fields of the cerebral cortex, early researchers [7] developed this connectionist model for mapping nonlinear behavior of static processes and for function approximation purposes. The basic RBFN structure consists of an input layer, a single hidden layer with a radial activation function and an output layer as shown in Figure 5.7.

In recent years, the usage of RBFN has been extended to a much wider range of applications involving dynamic systems, pattern classification,



**Figure 5.7:** Schematic representation of an RBF network

prediction and control. The network structure uses nonlinear transformations at its hidden layer (typical transfer functions for hidden functions are Gaussian curves), but uses linear transformations between the hidden and the output layers. The rationale behind this is that input spaces, cast nonlinearly into high-dimensional domains, are more likely to be linearly separable than those cast into low-dimensional ones. Unlike most FF neural networks, the connection weights between the input layer and the neuron units of the hidden layer for an RBFN are all equal to unity. The nonlinear transformations at the hidden layer level have the main characteristic of being symmetrical; they also attain their maximum at the function center, and generate positive values that decrease rapidly with distance from the center. As such they produce radial activation signals that are bounded and localized. The two parameters that characterize each activation function are known as the center and the width (normalization parameter). As is the case for other FF networks, a given RBFN has to learn its parameters during the training process. For an optimal performance of the network, the hidden layer nodes should span the training data input space. Too sparse or too overlapping functions may cause degradation of the network performance.

In general the form taken by an RBF function is given as

$$g_i(\mathbf{x}) = r_i \left( \frac{\|\mathbf{x} - \mathbf{v}_i\|}{\sigma_i} \right) \quad (5.14)$$

where  $\mathbf{x}$  is the input vector and  $\mathbf{v}_i$  is the vector denoting the center of the receptive field unit  $g_i$  with  $\sigma_i$  as its unit width parameter. The most widely used form of RBF is the Gaussian kernel function given by

$$g_i(\mathbf{x}) = \exp \left( \frac{-\|\mathbf{x} - \mathbf{v}_i\|^2}{2\sigma_i^2} \right) \quad (5.15)$$

The logistic function has also been used as a possible RBF candidate:

$$g_i(\mathbf{x}) = \frac{1}{1 + \exp(\|\mathbf{x} - \mathbf{v}_i\|^2/\sigma_i^2)} \quad (5.16)$$

As such, a typical output of an RBF network having  $n$  units in the hidden layer and  $r$  output units is given by:

$$o_j(\mathbf{x}) = \sum_{i=1}^n w_{ij} g_i(\mathbf{x}) \quad j = 1, \dots, r \quad (5.17)$$

where  $w_{ij}$  is the connection weight between the  $i$ -th receptive field unit and the  $j$ -th output, and  $g_i$  is the  $i$ -th receptive field unit.

### 5.3.2 Learning algorithm for RBF

The standard technique used to train an RBF network is the hybrid approach, which is a two-stage learning strategy. First, an unsupervised clustering

algorithm is used to extract the parameters of the radial basis functions, namely the width and the centers. This is followed by the computation of the weights of the connections between the output nodes and the kernel functions using a supervised least mean square algorithm. To tune even further the parameters of the networks, a supervised gradient-based algorithm is later applied and it makes use of some of the training patterns presented to the network. The reader may also wish to consult [4] for a detailed description of the algorithm.

In short, the training of the RBFN consists of two separate steps:

*Step 1: Train the RBFN layer to get the adaptation of centers and scaling parameters using the unsupervised training.*

The centers ( $\mathbf{v}_i$ ) of radial basis functions for each hidden node define input vectors causing maximal activation of these units. The widths ( $\sigma_i$ ) of the radial basis functions of each hidden node determine the radii of the areas of the input space around the centers where activations of these nodes are significant. To determine the centers of the radial basis functions, a number of unsupervised training procedures could be used. These include but are not restricted to the *k-means*-based method, the *maximum likelihood estimate*-based technique, the *standard deviations*-based approach, and the *self-organizing map* method. This step is very important for the construction of the RBFN, as the accurate knowledge of  $\mathbf{v}_i$  and  $\sigma_i$  has a major impact on the performance of the network.

*Step 2: Adapt the weights of the output layer using the supervised training algorithm.*

Once the centers and the widths of the radial basis functions are obtained, the next stage of the training begins. For this we can use supervised learning-based techniques such as the least-squares method or the gradient method to update the weights between the hidden layer and the output layer. Because the weights exist only between the hidden layer and the output layer, it is easier to compute the weight matrix for the RBFN than for the MLP network.

In the case where the RBFN is used for interpolation purposes, we can use the inverse or pseudo-inverse method to calculate the weight matrix. In this case, the radial basis functions  $g_{ij}$  will have as center the training data itself. For example, if we use the Gaussian kernel as the radial basis function and there are  $n$  input data, we have:

$$G = [\{g_{i,j}\}] \quad (5.18)$$

$$\text{where } g_{ij} = \exp\left(\frac{-\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma_i^2}\right), \quad (i = 1, \dots, n), (j = 1, \dots, n)$$

Now we have:

$$D = GW \quad (5.19)$$

where  $D$  is the desired output of the training data.  $G$  comes from equation (5.18).

If  $G^{-1}$  exists, we get:

$$W = G^{-1}D \quad (5.20)$$

In practice, however,  $G$  may be ill-conditioned (close to singularity) or may even be a non-square matrix (if the number of radial basis functions is less than the number of training data). Then  $W$  becomes expressed as:

$$W = G^+D \quad (5.21)$$

where  $G^+$  denotes the pseudo-inverse matrix of  $G$ , which can be defined as

$$G^+ = (G^T G)^{-1} G^T \quad (5.22)$$

Once the weight matrix has been obtained, all elements of the RBFN are now determined and the network could operate on the task it has been designed for.

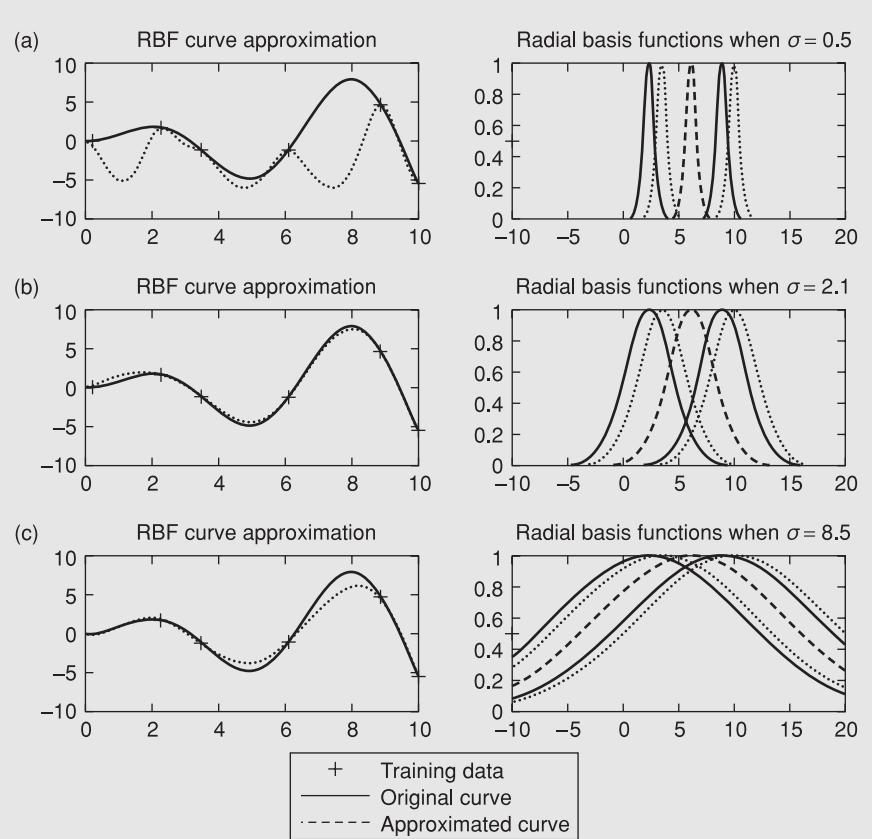
As can be seen, the two learning steps use different types of learning algorithms (unsupervised and supervised, respectively), which has led researchers to call the RBFN learning process a hybrid learning system.

### Example 5.4

#### *Approximation power of the radial basis function network*

We use here the same function as the one used in the MLP example. The RBF network is composed here of six radial functions and each has its center at a training input data. Three width parameters are used here: 0.5, 2.1, and 8.5. The results of simulation show that the width of the function is of major importance.

As Figure 5.8 shows, a smaller width value (relatively to the spacing in between the pattern) does not seem to provide for a good interpolation of the function in between sample data, as shown in Figure 5.8a. A larger width value ( $\Delta = 2.1$ ) provides a much better result and the approximation by RBF matches almost exactly the original curve. This particular width value seems to provide the network with an adequate interpolation property, as shown in Figure 5.8b. A larger width value ( $\Delta = 8.5$ ) seems to be inadequate for this particular case, given that a lot of information is being lost when the ranges of the radial functions are further away from the original range of the function. This is shown clearly in Figure 5.8c. One might conclude here that some testing is required before one gets the optimum approximation capability of the network. Of course there are algorithms which provide for the best choices of width parameters for each of the radial functions using an unsupervised type of learning as explained earlier.



**Figure 5.8 a, b, c:** Approximation of the function  $f(x)$  by an RBF network

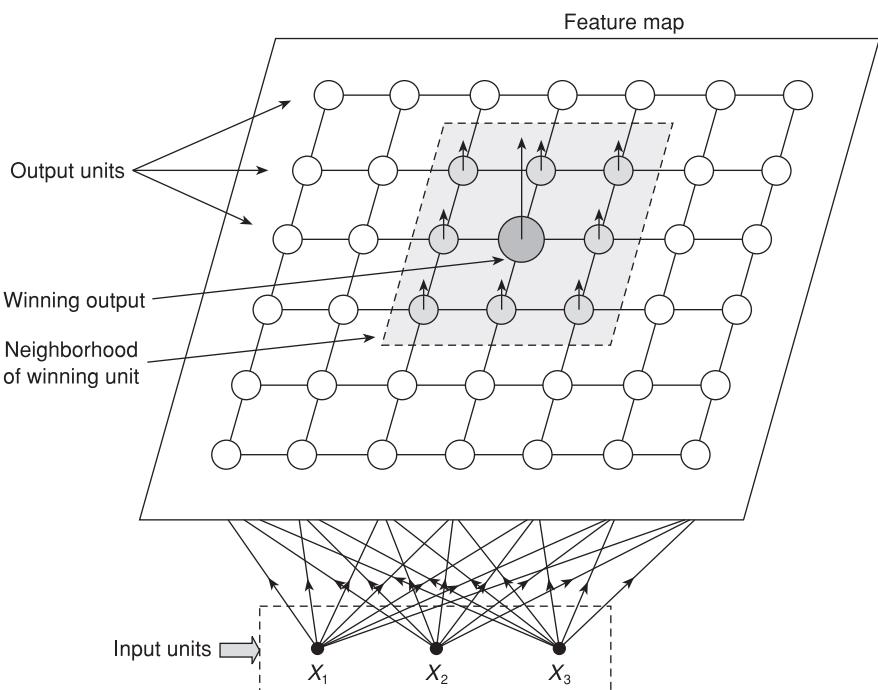
### 5.3.3 Applications

Known to have universal approximation capabilities, good local structures and efficient training algorithms, RBFN have often been used for nonlinear mapping of complex processes and for solving a wide range of classification problems. They have been used as well for control systems, audio and video signal processing, and pattern recognition, to name a few applications. They have also been recently used for chaotic time series prediction [8], with particular application to weather and power load forecasting. Generally, RBF networks have an undesirably high number of hidden nodes, but the dimension of the space can be reduced by careful planning of the network.

## 5.4 Kohonen's self-organizing network

### 5.4.1 Topology

The Kohonen self-organizing network (KSON), also known as the Kohonen self-organizing map (KSOM) or self-organizing map (SOM), belongs to the class of unsupervised learning networks. This means that the network, unlike other forms of supervised learning-based networks, updates its weighting parameters without the need for a performance feedback from a *teacher* or a network trainer. One major feature of this network is that the nodes distribute themselves across the input space to recognize groups of similar input vectors, while the output nodes compete among themselves to be fired one at a time in response to a particular input vector. This process is known as competitive learning. When suitably trained, the network produces a low-dimension representation of the input space that preserves the ordering of the original structure of the network. This implies that two input vectors with similar pattern characteristics excite two physically close layer nodes. In other words, the nodes of the KSON can recognize groups of similar input vectors. This generates a topographic mapping of the input vectors to the output layer, which depends primarily on the pattern of the input vectors and results in dimensionality reduction of the input space. This is done through an appropriate conversion of feature space. A schematic representation of a typical KSON with a 2-D output configuration is shown in Figure 5.9.



**Figure 5.9:** Schematic representation of the Kohonen self-organizing network

## 5.4.2 Learning algorithm

The learning here permits the clustering of input data into a smaller set of elements having similar characteristics (features). It is based on the competitive learning technique also known as the ‘*winner take all*’ strategy. Let us presume that the input pattern is given by the vector  $\mathbf{x}$  and let us denote by  $\mathbf{w}_{ij}$  the weight vector connecting the pattern input elements to an output node with coordinates provided by indices  $i$  and  $j$ . In most algorithm implementations, the weight vectors are normalized to unit length. Let us also denote  $N_c$  as being the neighborhood around the winning output candidate, which has its size decreasing at every iteration of the algorithm until convergence occurs. The steps of the learning algorithm are summarized as follows:

*Step 1:* Initialize all weights to small random values. Set a value for the initial learning rate  $\alpha$  and a value for the neighborhood  $N_c$ .

*Step 2:* Choose an input pattern  $\mathbf{x}$  from the input data set.

*Step 3:* Select the winning unit  $c$  (the index of the best matching output unit) such that the performance index  $I$  given by the Euclidian distance from  $\mathbf{x}$  to  $\mathbf{w}_{ij}$  is minimized:

$$I = \|\mathbf{x} - \mathbf{w}_c\| = \min_{ij} \|\mathbf{x} - \mathbf{w}_{ij}\|$$

*Step 4:* Update the weights according to the global network updating phase from iteration  $k$  to iteration  $k + 1$  as:

$$\mathbf{w}_{ij}(k+1) = \begin{cases} \mathbf{w}_{ij}(k) + \alpha(k)[\mathbf{x} - \mathbf{w}_{ij}(k)], & \text{if } (i, j) \in N_c(k) \\ \mathbf{w}_{ij}(k) & \text{if } (i, j) \notin N_c(k) \end{cases}$$

where  $\alpha(k)$  is the adaptive learning rate (strictly positive value smaller than unity) and  $N_c(k)$  is the neighborhood of the unit  $c$  at iteration  $k$ .

*Step 5:* The learning rate and the neighborhood are decreased at every iteration according to an appropriate scheme. For instance, Kohonen suggested a shrinking function in the form of  $\alpha(k) = \alpha(0)(1 - k/T)$ , with  $T$  being the total number of training cycles and  $\alpha(0)$  the starting learning rate bounded by one. As for the neighborhood, several researchers suggested an initial region with the size of half of the output grid which is shrinking according to an exponentially decaying behavior.

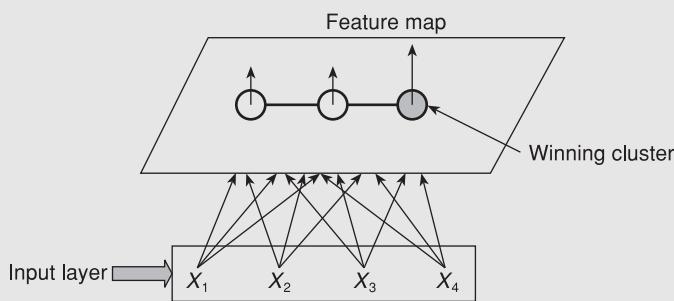
*Step 6:* The learning scheme continues until a sufficient number of iterations has been reached or until each output reaches a threshold of sensitivity with respect to a portion of the input space.

### Example 5.5

A Kohonen self-organizing map is used to cluster four vectors given by:  $(1, 1, 1, 0)$ ;  $(0, 0, 0, 1)$ ;  $(1, 1, 0, 0)$ ;  $(0, 0, 1, 1)$ . The maximum numbers of clusters to be formed is  $m = 3$  (Figure 5.10). Suppose the learning rate (geometric decreasing) is given by:

$$\begin{cases} \alpha(0) = 0.3 \\ \alpha(t+1) = 0.2\alpha(t) \end{cases}$$

With only three clusters available and the weights of only one cluster updated at each step (i.e.,  $N_c = 0$ ), find the weight matrix. Use one single epoch of training.



**Figure 5.10:** Structure of the neural network of Example 5.5

*Step 0:* The initial weight matrix is:

$$\mathbf{w} = \begin{bmatrix} 0.2 & 0.4 & 0.1 \\ 0.3 & 0.2 & 0.2 \\ 0.5 & 0.3 & 0.5 \\ 0.1 & 0.1 & 0.1 \end{bmatrix}$$

Initial radius:  $N_c = 0$

Initial learning rate:  $\alpha(0) = 0.3$

*Step 1:* Begin training.

*Step 2 (1):* For the first input vector  $\mathbf{x}_1 = (1, 1, 1, 0)$ , do steps 3–5.

*Step 3:*

$$I(1) = (1 - 0.2)^2 + (1 - 0.3)^2 + (1 - 0.5)^2 + (0 - 0.1)^2 = 1.39$$

$$I(2) = (1 - 0.4)^2 + (1 - 0.2)^2 + (1 - 0.3)^2 + (0 - 0.1)^2 = 1.5$$

$$I(3) = (1 - 0.1)^2 + (1 - 0.2)^2 + (1 - 0.5)^2 + (0 - 0.1)^2 = 1.71$$

*Step 4:* The input vector is closest to output node 1. Thus node 1 is the winner. The weights for node 1 should be updated.

*Step 5:* The weights on the winning unit are updated:

$$\begin{aligned}\mathbf{w}_1^{\text{new}} &= \mathbf{w}_1^{\text{old}} + \alpha(\mathbf{x} - \mathbf{w}_1^{\text{old}}) \\ &= (0.2, 0.3, 0.5, 0.1) + 0.3(0.8, 0.7, 0.5, 0.9) \\ &= (0.44, 0.51, 0.65, 0.37)\end{aligned}$$

$$\mathbf{w} = \begin{bmatrix} 0.44 & 0.4 & 0.1 \\ 0.51 & 0.2 & 0.2 \\ 0.65 & 0.3 & 0.5 \\ 0.37 & 0.1 & 0.1 \end{bmatrix}$$

*Step 2 (2):* For the second input vector  $\mathbf{x}_2 = (0, 0, 0, 1)$ , do steps 3–5.

*Step 3:*

$$\begin{aligned}I(1) &= (0 - 0.44)^2 + (0 - 0.51)^2 + (0 - 0.65)^2 + (1 - 0.37)^2 = 1.2731 \\ I(2) &= (0 - 0.4)^2 + (0 - 0.2)^2 + (0 - 0.3)^2 + (1 - 0.1)^2 = 1.1 \\ I(3) &= (0 - 0.1)^2 + (0 - 0.2)^2 + (0 - 0.5)^2 + (1 - 0.1)^2 = 1.11\end{aligned}$$

*Step 4:* The input vector is closest to output node 2. Thus node 2 is the winner. The weights for node 2 should be updated.

*Step 5:* The weights on the winning unit are updated:

$$\begin{aligned}\mathbf{w}_2^{\text{new}} &= \mathbf{w}_2^{\text{old}} + \alpha(\mathbf{x} - \mathbf{w}_2^{\text{old}}) \\ &= (0.4, 0.2, 0.3, 0.1) + 0.3(-0.4, -0.2, -0.3, 0.9) \\ &= (0.28, 0.14, 0.21, 0.37)\end{aligned}$$

$$\mathbf{w} = \begin{bmatrix} 0.44 & 0.28 & 0.1 \\ 0.51 & 0.14 & 0.2 \\ 0.65 & 0.21 & 0.5 \\ 0.37 & 0.37 & 0.1 \end{bmatrix}$$

*Step 2 (3):* For the third input vector  $\mathbf{x}_3 = (1, 1, 0, 0)$ , do steps 3–5.

*Step 3:*

$$\begin{aligned}I(1) &= (1 - 0.44)^2 + (1 - 0.51)^2 + (0 - 0.65)^2 + (0 - 0.37)^2 = 1.1131 \\ I(2) &= (1 - 0.28)^2 + (1 - 0.14)^2 + (0 - 0.21)^2 + (0 - 0.37)^2 = 1.439 \\ I(3) &= (1 - 0.1)^2 + (1 - 0.2)^2 + (0 - 0.5)^2 + (0 - 0.1)^2 = 1.71\end{aligned}$$

*Step 4:* The input vector is closest to output node 1. Thus node 1 is the winner. The weights for node 1 should be updated.

*Step 5:* The weights on the winning unit are updated:

$$\begin{aligned}\mathbf{w}_1^{\text{new}} &= \mathbf{w}_1^{\text{old}} + \alpha(\mathbf{x} - \mathbf{w}_1^{\text{old}}) \\ &= (0.44, 0.51, 0.65, 0.37) + 0.3(0.56, 0.49, -0.65, -0.37) \\ &= (0.608, 0.657, 0.455, 0.259)\end{aligned}$$

$$\mathbf{w} = \begin{bmatrix} 0.608 & 0.28 & 0.1 \\ 0.657 & 0.14 & 0.2 \\ 0.455 & 0.21 & 0.5 \\ 0.259 & 0.37 & 0.1 \end{bmatrix}$$

*Step 2 (4):* For the third input vector  $\mathbf{x}_4 = (0, 0, 1, 1)$ , do steps 3–5.

*Step 3:*

$$\begin{aligned}I(1) &= (0 - 0.608)^2 + (0 - 0.657)^2 + (1 - 0.455)^2 + (1 - 0.259)^2 = 1.647419 \\ I(2) &= (0 - 0.28)^2 + (0 - 0.14)^2 + (1 - 0.21)^2 + (1 - 0.37)^2 = 1.119 \\ I(3) &= (0 - 0.1)^2 + (0 - 0.2)^2 + (1 - 0.5)^2 + (1 - 0.1)^2 = 1.11\end{aligned}$$

*Step 4:* The input vector is closest to output node 3. Thus node 3 is the winner. The weights for node 3 should be updated.

*Step 5:* The weights on the winning unit are updated:

$$\begin{aligned}\mathbf{w}_3^{\text{new}} &= \mathbf{w}_3^{\text{old}} + (\mathbf{x} - \mathbf{w}_3^{\text{old}}) \\ &= (0.1, 0.2, 0.5, 0.1) + 0.3(-0.1, -0.2, 0.5, 0.9) \\ &= (0.07, 0.14, 0.65, 0.37)\end{aligned}$$

$$\mathbf{w} = \begin{bmatrix} 0.608 & 0.28 & 0.07 \\ 0.657 & 0.14 & 0.14 \\ 0.455 & 0.21 & 0.65 \\ 0.259 & 0.37 & 0.37 \end{bmatrix}$$

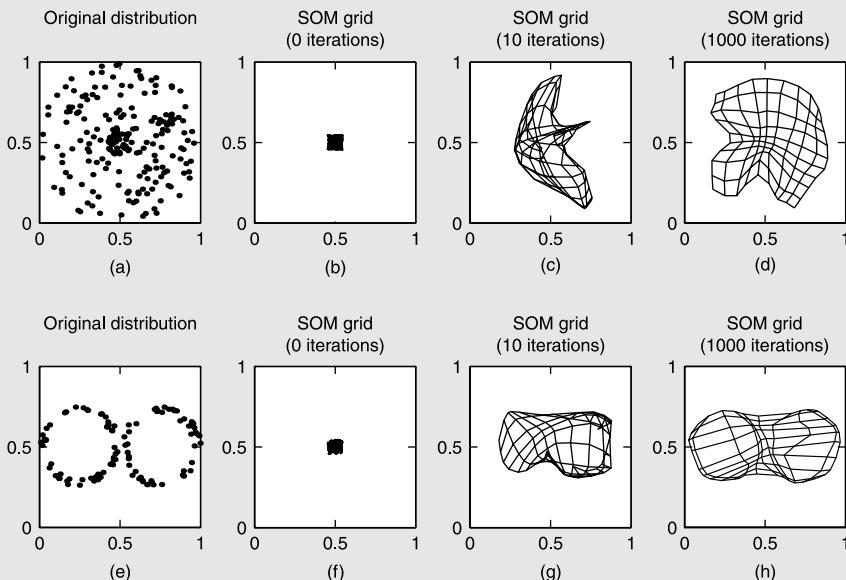
*Step 6:* Epoch 1 is complete. Reduce the learning rate:

$\alpha(t+1) = 0.2\alpha(t) = 0.2(0.3) = 0.06$  and repeat from the start for new epochs until  $\Delta\mathbf{w}_j$  becomes steady for all input patterns or the error is within a tolerable range.

## Example 5.6

### *Self-organization of random data constrained within a spatial domain*

To illustrate the principle of self-organization in SOM networks, we provide here an example showing how clusters of data are being self-organized in a reduced dimension set using competitive learning. In Figure 5.11a, we provide as the initial space of data a random set of patterns located uniformly within a disk of radius 0.5. Inside the disk, we add another set of data uniformly distributed within a disk of radius 0.1. After 1000 iterations, the SOM was able to organize the data into a low number of clusters as shown in Figure 5.11 (a–d). The same was applied to another space of input patterns composed of two loosely connected tauruses; each has an inner radius of 0.2 and an outer radius of 0.25. After a thousand iterations, the data input have arranged themselves into a grid of clusters as shown in Figure 5.11 (e–h).



**Figure 5.11 (a–d):** Self-organization of random data constrained within a disk of radius 0.1

**Figure 5.11 (e–h):** Self-organization of random data located within two interconnected tauruses

### 5.4.3 Applications

Given their self-organizing capabilities based on the competitive learning rule, KSONs have been used extensively for clustering applications such as in speech recognition, vector coding, and texture segmentation. They have also been used in robotics applications and for designing nonlinear controllers.

A variety of KSONs could be applied to different applications using the different parameters of the network, which are: the neighborhood size and shape (circular, square, diamond), the learning rate decaying behavior, and the dimensionality of the neuron array (1-D, 2-D or  $n$ -D).

## 5.5 The Hopfield network

### 5.5.1 Topology

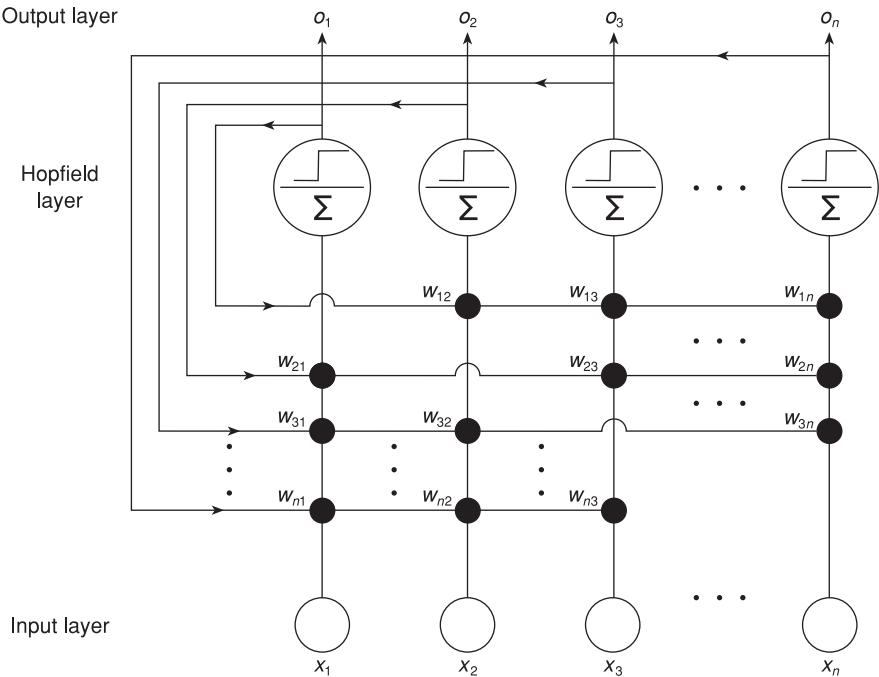
A very special and interesting case of recurrent (feedback) topology is the Hopfield network. It was the pioneering work of Hopfield in the early 1980s that led the way for designing neural networks with feedback paths and dynamics. The work of Hopfield is seen by many as the starting point for the implementation of associative (content addressable) memory by using a special structure of recurrent neural networks. The associative memory concept means simply that the network is able to recognize newly presented (noisy or incomplete) patterns using an already stored “complete” version of that pattern. We say that the new pattern is “attracted” to the stable pattern already stored in the network memories. This stems from what Hopfield defines as a physical system [9]: *Any physical system whose dynamics in phase space is dominated by a substantial number of locally stable states to which it is attracted can therefore be regarded as a general content-addressable memory.* In other words, this could be stated as having the network represented by an energy function that keeps decreasing until the system has reached stable status. The general structure of the Hopfield network, also shown in Figure 5.12, is made up of a number of processing units configured in one single layer (besides the input and the output layers) with symmetrical synaptic connections, i.e.,  $w_{ij} = w_{ji}$ .

In the original work of Hopfield, the output of each unit can take a binary value (either 0 or 1) or a bipolar value (either -1 or 1). This value is fed back to all the input units of the network except to the one corresponding to that output. Let us suppose here that the state of the network with dimension  $n$  ( $n$  neurons) takes bipolar values. The activation rule for each neuron is then provided by the following:

$$o_i = \text{sign}\left(\sum_{\substack{i=1 \\ i \neq j}}^n w_{ij} o_j - \theta_i\right) = \begin{cases} +1 & \text{if } \sum_{i \neq j} w_{ij} o_j > \theta_i \\ -1 & \text{if } \sum_{i \neq j} w_{ij} o_j < \theta_i \end{cases} \quad (5.23)$$

with  $o_i$  denoting the output of the current processing unit (Hopfield neuron) and  $\theta_i$  its threshold value. This also corresponds to the output of the signum function. To accomplish the autoassociative behavior of the network according to his physical view of a dynamical system, Hopfield used an energy function for the network given by:

$$E = -\frac{1}{2} \sum_{i \neq j} \sum_{i \neq j} w_{ij} o_i o_j + \sum_i o_i \theta_i \quad (5.24)$$



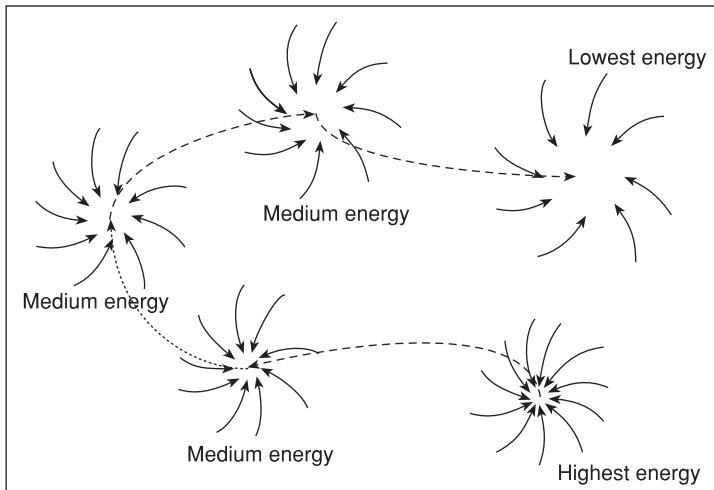
**Figure 5.12:** Schematic representation of a Hopfield network

To ensure stability of the network,  $E$  is defined in such a way as to decrease monotonically with variation of the output states until a minimum is attained. This could be readily noticed from the expression relating the variation of  $E$  with respect to the output states' variation. In fact it could be easily shown that

$$\Delta E = -\Delta o_i \left( \sum_{j \neq i} w_{ij} o_j - \theta_i \right) \quad (5.25)$$

Expression (5.25) shows indeed that the energy function  $E$  of the network continues to decrease until it settles by reaching a local minimum. This also translates into a monotonically decreasing behavior of the energy. Because of this interesting characteristic of the network, it can be shown that any new pattern input presented to the network will get attracted to the locally stable regions of the network. These regions correspond exactly to what the system has memorized during the learning stage, and hence the connotation of autoassociative memory used to represent the behavior of the Hopfield network (Figure 5.13), since the network associates with each fundamental memory other patterns that are corrupted or are a noisy version of the stored patterns.

A few years after his original contribution, Hopfield extended the capabilities of his network to involve possible continuous states. This work is summarized in detail in a seminal paper he presented in 1984 at the National Academy of Sciences [9].



**Figure 5.13:** Typical transition of patterns from high energy levels to lower energy levels. Fundamental patterns will occupy certain energy levels within the energy space

### 5.5.2 Learning algorithm

The learning algorithm for the Hopfield network is based on the so-called Hebbian learning rule. This is one of the earliest procedures designed for carrying out supervised learning. It is based on the idea that when two units are simultaneously activated, their interconnection weight increase becomes proportional to the product of their two activities. The Hebbian learning rule, also known as the outer product rule of storage, as applied to a set of  $q$  presented patterns  $\mathbf{p}_k (k = 1, \dots, q)$  each with dimension  $n$  ( $n$  denotes the number of neuron units in the Hopfield network), is expressed as:

$$w_{ij} = \begin{cases} \frac{1}{n} \sum_{k=1}^q p_{kj} p_{ki} & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases} \quad (5.26)$$

The weight matrix  $\mathbf{W} = \{w_{ij}\}$  could also be expressed in terms of the outer product of the vector  $\mathbf{p}_k$  as:

$$\mathbf{W} = \{\mathbf{w}_{ij}\} = \frac{1}{n} \sum_{k=1}^q \mathbf{p}_k \mathbf{p}_k^T - \frac{q}{n} \mathbf{I} \quad (5.27)$$

The ratio  $(1/n)$  in (5.26) and (5.27) is used for computational convenience and could be disposed of as in example 5.7 below. The different learning stages of a Hopfield network are summarized next:

**Step 1 (storage):** The first stage in the learning algorithm is to store the patterns through establishing the connection weights using equation (5.26) or (5.27). Each of the  $q$  patterns presented is a vector of bipolar elements (+1 or -1). These patterns are also called fundamental memories, given that the connection

weights have now become a fixed entity within the network. This reflects the stable status of the network to which newly presented patterns should get “attracted.”

*Step 2 (initialization):* The second stage is initialization and it consists in presenting to the network an unknown pattern  $\mathbf{u}$  with the same dimension as the fundamental patterns. Every component of the network outputs at the initial iteration cycle is set as

$$\mathbf{o}(0) = \mathbf{u} \quad (5.28)$$

*Step 3 (retrieval 1):* Each one of the component  $o_i$  of the output vector  $\mathbf{o}$  is updated from cycle  $l$  to cycle  $l + 1$  according to the iteration formulae:

$$o_i(l+1) = \text{sgn}\left(\sum_{j=1}^n w_{ij} o_j(l)\right) \quad (5.29)$$

This process is known as asynchronous updating (where a single neuron is selected randomly and is activated according to equation (5.29) of step 3). The process continues until no more changes are made and convergence occurs. At this stage the iteration is stopped and the output vector obtained matches best the fundamental pattern closest to it.

*Step 4 (retrieval 2):* Continue the process for other presented unknown patterns by starting again from step 2.

### Example 5.7

We need to store a fundamental pattern (memory) given by the vector  $B = [1 \ 1 \ 1 \ -1]^T$  in a four-node binary Hopfield network. We will show that all possible 16 states of the network will be attracted to the fundamental memory given by  $B$  or to its complement  $L = [-1 \ -1 \ -1 \ 1]^T$ .

Let us presume that all threshold parameters  $\theta_i$  are equal to zero. The weight matrix is given by equation (5.26), where the ratio ( $1/n = 1/4$ ) was discarded, and corresponds to:

$$W = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}$$

Using the energy equation we compute the energy value for every state (each state is coded with entries  $o_i$ , where  $o_i = 1$  or  $-1$  for each  $i = 1, 2, 3, 4$ ). This is given in Table 5.1.

Looking at the energy levels, we find that two potential attractors emerge: the original fundamental pattern coded by  $[1, 1, 1, -1]^T$  and its complement given by  $[-1, -1, -1, 1]^T$ . Both of them have the lowest energy value given by “-6”.

**Table 5.1:** All possible network states, their code, and their corresponding energies

State	Code	Energy
A	1 1 1 1	0
B	1 1 1 -1	-6
C	1 1 -1 -1	0
D	1 1 -1 1	2
E	1 -1 -1 1	0
F	1 -1 -1 -1	2
G	1 -1 1 -1	0
H	1 -1 1 1	2
I	-1 -1 1 1	0
J	-1 -1 1 -1	2
K	-1 -1 -1 -1	0
L	-1 -1 -1 1	-6
M	-1 1 -1 1	0
N	-1 1 -1 -1	2
O	-1 1 1 -1	0
P	-1 1 1 1	2

At the retrieval stage, we use

$$o_i(l+1) = \text{sgn} \left( \sum_{\substack{i=1 \\ i \neq j}}^n w_{ij} o_j - \theta_i \right) \quad (5.30)$$

In this example we presume that all  $\theta_i = 0$ , and we update the state asynchronously using equation (5.30). Updating the state asynchronously means that for every state presented we activate one neuron at a time, which leads to the state transitions in Table 5.2. We notice that all states either remain at their current energy level or move to lower energy levels as shown in Table 5.2 and Figure 5.14.

Let us analyze briefly some of the transitions:

- (1) When the input state is  $[1, 1, 1, 1]^T$ , which is the state A with energy “0”, we find that only the fourth transition contributes to a change in status. As such, the state A transits to  $[1, 1, 1, -1]^T$ , which is the state B, representing the fundamental pattern with the lowest energy value “-6”.
- (2) When the input state is  $[1, 1, -1, 1]^T$ , which is state D with energy value “2”, it could transit a few times to end up at state L after being updated asynchronously. In fact we find that all of its four bits  $o_1, o_2, o_3, o_4$  can be updated. Updating the first bit  $o_1$ , the state becomes  $[-1, 1, -1, 1]^T$ , which is the state M with energy level “0”. Updating bit  $o_2$ , the state transits to  $[1, -1, -1, 1]^T$ , which is the state L having the energy level “-6”. Updating bit  $o_3$ , the state remains at  $[-1, -1, -1, 1]^T$ . Updating bit  $o_4$ , the state remains unchanged as L. Throughout this process, the energy has decreased from level “2” to level “0” to level “-6”. We also

Table 5.2: Asynchronous state transitions

State/energy	Code	Transition 1 ( $o_1$ )	Transition 2 ( $o_2$ )	Transition 3 ( $o_3$ )	Transition 4 ( $o_4$ )
A/(0)	1 1 1 1	1 1 1 (A)	1 1 1 (A)	1 1 1 (A)	1 1 1 (B)
B/(-6)	1 1 1 -1	1 1 1 (B)			
C/(0)	1 1 -1 -1	1 1 -1 (C)	1 1 -1 (C)	1 1 -1 (C)	1 1 -1 (B)
D/(2)	1 1 -1 1	-1 1 -1 (W)	-1 1 -1 (U)	-1 -1 1 (U)	-1 -1 1 (U)
E/(0)	1 -1 -1 1	-1 -1 -1 (U)	-1 -1 -1 (U)	-1 -1 1 (U)	-1 -1 1 (U)
F/(2)	1 -1 -1 -1	-1 -1 -1 (K)	-1 -1 -1 (K)	-1 -1 -1 (K)	-1 -1 -1 (U)
G/(0)	1 -1 1 -1	1 -1 1 (G)	1 -1 1 (B)	1 1 1 (B)	1 1 1 (B)
H/(2)	1 -1 1 1	-1 1 1 (O)	-1 1 1 (O)	-1 1 1 (O)	-1 1 1 (U)
I/(0)	-1 1 1 1	-1 1 1 (O)	-1 1 1 (O)	-1 1 1 (O)	-1 1 1 (U)
J/(2)	-1 1 -1 1	1 -1 -1 (G)	1 1 1 (B)	1 1 1 (B)	1 1 1 (B)
K/(0)	-1 -1 -1 1	-1 -1 -1 (K)	-1 -1 -1 (K)	-1 -1 -1 (K)	-1 -1 -1 (U)
L/(-6)	-1 -1 1 -1	-1 -1 1 (U)			
M/(0)	-1 1 -1 1	-1 1 -1 (W)	-1 -1 1 (U)	-1 -1 1 (U)	-1 -1 1 (U)
N/(2)	-1 1 -1 -1	1 1 -1 (C)	1 1 -1 (C)	1 1 -1 (B)	1 1 -1 (B)
O/(0)	-1 1 1 -1	1 1 1 (B)	1 1 1 (A)	1 1 1 (A)	1 1 1 (B)
P/(2)	-1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1

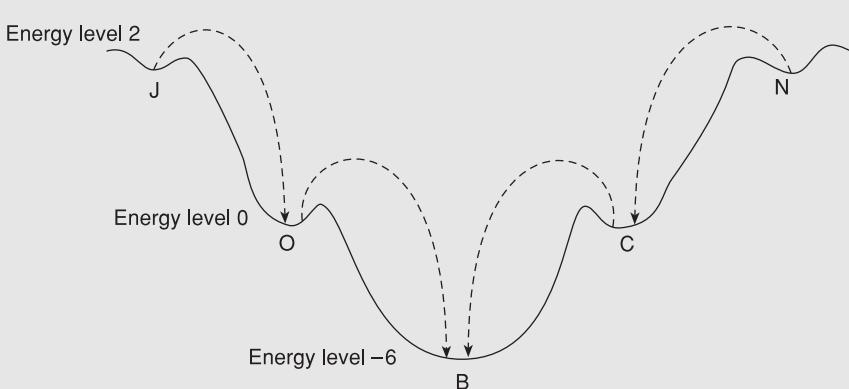


Figure 5.14: Transition of state J and N from high energy levels to lower energy levels

know that the state  $D$  can transit to other states before settling at  $B$  or  $L$ . This depends on which bit is being updated first. If the state  $D$  transits to state  $A$  or  $C$ , it will continue the updating and ultimately transits to the fundamental state  $B$ , which has the energy “ $-6$ ”. If the state  $D$  transits to state  $E$  or  $M$ , it will continue the updating and ultimately transits to state  $L$ , which also has the lowest energy “ $-6$ ”. This is illustrated in the transition diagram of Figure 5.15.

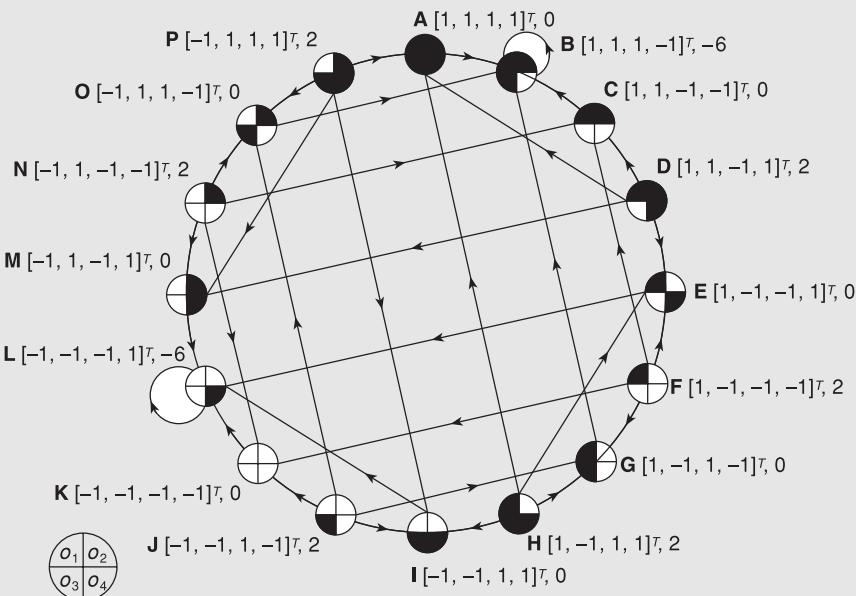


Figure 5.15: The state transition diagram where each node is characterized by its vector state and its energy level

- (3) When the input state is the fundamental pattern  $B$  represented by  $[1, 1, -1]^T$ , there is no change of the energy level and no transition occurs to any other state. It is in its stable state because this state has the lowest energy.
- (4) When the input state is  $[-1, -1, -1, 1]^T$ , which happens to be the complement of the fundamental pattern  $B$ , given by the state  $L$ , we find that its energy level is the same as  $B$  and hence it is another stable state. This is usually the case: every complement of a fundamental pattern is a fundamental pattern itself. This is due to the fact that the Hamming distance is the same for both the fundamental pattern and its complement. This means that the Hopfield network has the ability to remember the fundamental memory and its complement.

### 5.5.3 Applications of Hopfield networks

Given their autoassociative memories capabilities, Hopfield networks have been used extensively for information retrieval and for pattern and speech recognition. They have been used as well to solve optimization problems. In fact, by making the energy function of a Hopfield network equivalent to a certain performance index or an objective function of a given process to be minimized, the configuration at which the network finally settles indeed represents the solution to the original optimization problem [10]. The Hopfield network was also used for solving combinatorial optimization problems such as the travelling salesman problem [11]. As with other types of networks, the Hopfield network, despite many of its features and retrieval capabilities, has its own limitations. They are mostly related to the limited stable-state storage capacity of the network and its possible convergence to an incorrect stable state when non-fundamental patterns are presented.

Hopfield estimated roughly that a network with  $n$  processing units should allow for  $0.15n$  stable states. Many studies have been carried out recently to increase the capacity of the network without increasing much the number of processing units.

## 5.6 Industrial and commercial applications of ANN

There has been a large volume of literature written in the area of industrial applications of neural networks. We provide in this chapter a sample of these applications pertaining in particular to: process monitoring and control, power systems, robotics, communication systems, pattern recognition, and decision fusion. This is not meant to be a detailed description of the techniques involved, but rather an overview that would assist the reader in grasping the main ideas proposed by researchers and designers in the field of neural network applications during the past few years. A large number of references are provided here that would complement the material described in this chapter.

### 5.6.1 Neural networks for process monitoring and optimal control

The neural network computing approach to process monitoring is founded on the idea that it is possible to “learn” a computational model of the process in question. This is achieved by observing the system to see how the process evolves with time or in response to additional external action. The network builds a functional relationship which mimics the actual process, describing accurately how the system would evolve from any specific initial state either on its own or in response to additional external actions, the so-called control-actions. In the present scheme of things, optimality is attained with the use of a mode of evolutionary programming consisting of guided evolutionary programming with simulating annealing (GESA). More generally, evolutionary programming is another root of computational paradigms, which are sometimes collectively referred to as the *technologies of computational intelligence* ([12], [13], [14], [15], and [16]). GESA together with feedforward nets can be used for optimal path planning and optimal control ([17] and [18]).

The primary task of process monitoring is to confirm that the process is proceeding as intended. Demarcation of the responsibilities of the process monitoring function is not precise, and there can be considerable overlap between the diagnostics and simple process monitoring. Neural network process monitoring differs from the charting practice of conventional statistical quality control (SQC) in that an entirely new dimension is added. In SQC, reported parameter values are charted and the variation of these with time is also charted. All that is still possible with the use of neural nets, but what is additionally available are measures of deviation of estimated values from the actually reported values. That provides a new type of information for process monitoring.

With the use of an accurately trained neural network model of the process, it is possible to take a set of system-state sensor readings as input to the model and predict the next step value of the system state. Continued agreement between actual values and the predicted values would indicate that the process is proceeding well. In real-world systems, processes are not completely deterministic. The training set data may contain noise, perhaps a great deal of noise at some instances. The extent of that is known from values of the training set system error and the validation set system error. Therefore, as monitoring proceeds, a record is kept of the root mean square error evaluated over short spans of time. It is the characteristics of the actual individual errors of the records of the average errors that indicate whether or not something has gone wrong.

### 5.6.2 Neural networks in semiconductor manufacturing processes

Because of the large number of steps involved in a semiconductor manufacturing process, maintaining product quality in an Integrated Circuits (IC) manufacturing facility requires strict control of literally hundreds or even thousands of process variables. The interdependent issues of high yield, high

quality, and low cycle time have been addressed in part by the ongoing development of several critical capabilities in state-of-the-art computer integrated manufacturing of integrated circuits (IC-CIM) systems, in-situ process monitoring, process/equipment modeling, real-time closed-loop process control, and equipment malfunction diagnosis. The emphasis on each of these activities is to increase throughput and reduce yield loss by preventing potential misprocessing, but each presents significant engineering challenges in their effective implementation and development [19].

Recently, there has been an explosion in the use of artificial neural networks in various manufacturing applications [20] and the semiconductor manufacturing arena has been no exception to this trend. Neural networks have emerged as powerful techniques for assisting IC-CIM systems in performing process monitoring, modeling, control and diagnosis functions. Because of their inherent learning ability, adaptability, and robustness, neural nets have been used to solve problems that have heretofore resisted solutions by other more traditional methods. There are several neural network architectures and training algorithms eligible for manufacturing applications. Hopfield networks, for example, have been used for solving combinatorial optimization problems such as optimal scheduling [21]. However, the back-propagation (BP) algorithm is the most generally applicable and thus far most popular in semiconductor manufacturing. It has been demonstrated by several authors that if a sufficient number of hidden neurons are present, a three-layer BP network can encode any arbitrary input–output relationship [22]. The ability of neural networks to learn input–output relationships from limited data is quite beneficial in semiconductor manufacturing, where a plethora of highly nonlinear fabrication processes exist, and experimental data for process modeling are expensive to obtain. Several researchers have recently reported noteworthy successes in using neural networks to model the behavior of a few key fabrication processes ([23], [24], [25], and [26]). In doing so, the basic strategy is usually to perform a series of statistically designed characteristics experiments, and then to train BP neural networks to model the experimental data. The process characterization experiments typically consist of a factorial or reduced factorial exploration of the input parameter space, which may be subsequently augmented by a more advanced experimental design. Each set of input conditions in the design corresponds to a particular set of measured process responses. This input–output mapping is precisely what the neural network learns. In each of the mentioned works standard implementation of the BP algorithm has been employed to perform the process monitoring tasks. However, innovative modifications of standard BP has also been developed for certain other applications of semiconductor process modeling. Kim and May [27] used simulated annealing to enhance the model accuracy. Nami *et al.* [28], on the other hand, developed a semi-empirical model of the metal organic chemical vapor deposition process based on hybrid neural networks. The main idea was to overcome the shortcomings of having little insight into the underlying physical understanding of the process being modeled by incorporating partial knowledge regarding the first principle relationships inherent in the process being modeled.

Work has been done in optimization within the semiconductor manufacturing application. One approach used statistical methods to optimize the neural process models themselves; the goal of this approach is to determine the proper network structure and the set of BP learning parameters to minimize network training error, prediction error, and training time. The second approach to optimization focuses on using neural process models to optimize a given semiconductor fabrication process or determine specific process recipes to achieve a desired response ([29] and [30]). Recently neural networks have been used to monitor the variations in manufacturing process conditions in real time, as opposed to the off line statistical process control approach ([31], [32], [33], [34], and [35]). They have also been used in developing real-time closed-loop process control schemes, which use in-situ process sensors to make on line adjustments in process set points ([36], [37], and [38]). Neural networks have also been widely used in process monitoring and diagnosis ([39] and [40]).

### 5.6.3 Neural networks for power systems

In the last few years, there has been a surge in using artificial neural networks (ANN) in the area of power systems. Numerous implementations have been made in this regard ranging from generator load forecasting to nuclear power plant stabilization. In what follows, we enumerate a number of studies pertaining to utilizing ANN in the field of power systems. In their work [41], Weersooriya and El-Sharkawi designed multilayer perceptron neural networks for static security assessment (SSA) of a large-scale power system. The neural networks are trained to provide the power system operator with the security status of the plant for specific contingencies according to some pre-contingency variables. The neural networks' parallel processing and adaptive capabilities led to increasing the speed of the classification with satisfactory performance.

Hsu and Chen [42] have also investigated the use of an artificial neural network to adaptively tune power system stabilizers (PSS) in an on line manner. The neural network was shown to be efficient under a wide range of generator loading conditions. In [43], an ANN-based PSS was compared to a self-optimizing pole shifting adaptive PSS (APSS). It was also shown that the ANN-based PSS had the capability of efficiently learning to maintain the merits of the APSS while neglecting its disadvantages. In the same context of power plant stabilization, Farag *et al.* [44] proposed an optimized neuro-fuzzy power system stabilizer (NF PSS) to improve upon the transient response and dynamic stability of synchronous machines. Simulation results have shown the superiority of the NF PSS to conventional PSS in a multi-machine power system environment.

Kraft *et al.* [45] proposed a hybrid expert system composed of a fully connected recurrent neural network and a knowledge base to provide an on line supervisory system for a power plant system. The intelligent system diagnoses the plant's boiler status at any given instant of time. In case of abnormality, the hybrid expert system acts to bring the plant to its normal

working condition and alerts the operator to the problem. This fast deficiency detection and recovery system was shown to drastically improve the plant's reliability.

Amjadi and Ehsan [46] developed a new method, which takes advantage of an ANN to analyze the reliability of power systems. The use of an ANN in such a method has several advantages, such as higher accuracy, simple modeling, and lower computational cost compared to previous reliability analysis methods. In [47], El-Sayed used a multilayer perceptron neural network for reactive power optimization of interconnected power systems. The neural network is designed to generate the optimal voltages at the buses controlling in real time the reactive power for different loading conditions. The network's relative architectural simplicity, high processing speed, and ability to model nonlinear functions are the major factors behind the high performance of the suggested approach.

Artificial neural networks were also found to be quite useful in power systems electrical load forecasting. Srinivasan *et al.* [48] implemented a hybrid short-term electrical load forecasting model for use in a power system control center. The hybrid architecture is composed of a Kohonen self-organizing map, a multilayer perceptron, and a fuzzy expert system. The Kohonen self-organizing map uses an unsupervised learning algorithm to classify daily load patterns. The multilayer perceptron uses a supervised backpropagation method to train the network with the temperature–load relationship. The fuzzy expert system post-processes the information coming out from the neural networks under a prespecified set of linguistic *if-then* rules. To ensure the adaptiveness of the global system, the learning process is designed to operate in an on line manner. Such a design has been applied to a real-world working environment and it was shown to outperform existing conventional methods. Given the success registered so far in the field of applying ANN to power systems, it is expected that more algorithms will be developed in the future to further enhance the current schemes, and to allow them to be applied on an even wider scale.

#### 5.6.4 Neural networks in robotics

Due to the increase in modeling and control complexities of the new generation robotic systems, there has been a strong trend in applying tools of connectionist modeling (artificial neural networks) in the area of robotics. The ubiquitous presence of uncertainty and the high nonlinear time-varying dynamics of a typical robotic system have made it harder to design reliable control strategies that are based on conventional model-based control tools. It is the learning capability of ANNs coupled with the parallel processing and generalization capabilities that made possible the successful applications of ANNs in the control of robotic systems. In the following, several of these applications are discussed.

Cui and Shin [50] presented a thorough study on why and how neural networks can be utilized efficiently for tackling multiple robotic structures. Following an overview on the theoretical foundation, the authors illustrate

their suggested methodology by applying it to two real-life cooperative robotic problems. The results obtained clearly show the superiority of neural networks as compared with the conventional techniques previously used in solving the problems approached.

Shibata *et al.* [51, 52] used MLP networks to tune different levels of a robotic control hierarchical structure. The neural networks are used to decide on the long-term learning for strategic planning as well as the control action adjustments for the dynamic processes. The suggested structure is tested on object recognition, strategic planning, and position/force control modules of a real-life robotic system. The results of the experiments are compared to those of conventional control techniques and the ANN-based architecture is shown to be much more efficient in many aspects.

ANNs have also been used in excavation robots. Shi *et al.* [53] implemented a neural network-based system to handle the decision-making of the robot. The neural network makes use of its past performance history to adapt itself to the current robot environment on line. In addition to supervised learning-based neural networks, unsupervised learning-based networks, such as Kohonen's self-organizing maps, have also been used in modern robotic systems. Self-organizing methodologies have been mainly applied in the manipulator kinematic control problem to avoid computing kinematics parameters such as link lengths and structures [54, 55, 56, 57, 58, 59].

ANNs are developing a growing interest in the area of mobile robotic systems as well. The cruising control of a mobile robot is heavily dependent on the dynamics of the robot cart. Determining the exact dynamics of the robot may be extremely difficult in many cases. One way of overcoming such a problem is to use a neural network that automatically adapts itself to the current dynamics of the system through its learning ability. Such a technique has been successfully applied to new generation mobile robots [60, 61, 62]. ANNs have also been used in the navigation guidance systems of mobile robots. Meng and Picton [63] present a neural network-based approach to the robot navigation problem in a dynamically varying environment. The neural network adopted integrates the information collected from the different sensors to guide the robot to a collision-free path. Although the robot environment contains moving obstacles with unknown trajectories, simulations show the effectiveness of neural networks in tackling such a complex problem. Many other path-planning techniques that use neural networks have also been suggested in the literature [64, 65].

### 5.6.5 Neural networks in communications

In recent years, neural networks have been used extensively to solve a wide range of problems related to communication systems. Applications included but were not restricted to broadband networks, wireless networks, telephone calls, switching, networks topology design, and satellite systems.

In broadband networks (e.g., ATM networks), neural networks have been utilized to tackle problems related to connection admission control (CAC), congestion control, switching control, and traffic conformance and monitoring.

In [66], Yousef *et al.* have proposed a *connection admission controller* that consists of two independent modules coordinated by a decision-maker. In module I, multiple three-layered backpropagation neural networks were trained to calculate the bandwidth required per call on each virtual path using on line traffic measurements. In module II, a three-layered backpropagation neural network was used to calculate the capacity required for the combined virtual paths. The objective here is to maximize the link utilization by including the gain achieved from statistical multiplexing and by determining the unused capacity of the link. According to [66], the reported results prove that the neurocomputing approach can achieve more accurate results than other conventional techniques based on mathematical and simulation analysis.

A neural network-based *policing* mechanism for ATM networks was proposed in [67]. This mechanism is called a neural networks traffic enforcement mechanism (NNTEM). The main idea is to estimate the probability density function of the multimedia traffic using its count process. NNTEM utilized two backpropagation perceptrons, NN1 and NN2. NN1 is trained with ideal non-violating traffic patterns, and is used to estimate the *probability* of a certain amount of non-violating traffic during their sampling time. NN2 is trained using patterns to learn about the violating traffic. An error signal is generated by comparing the output of NN1 and NN2. This signal is a prediction indication that violations may occur. The final action of the NNTEM controller is a signal used to drop violating cells.

Park *et al.* [68] proposed an Omega network based on a high throughput ATM *switch structure* with bypass queues. The switch control is implemented using a neural network to utilize the potential of the switch. A continuous Hopfield neural network is used to control the cell scheduling and multiplexing in the switch. The neurons in every buffer set form a *winner-take-all* circuit where the neurons have inhibitory connections among them with identical connection weights. According to [68], the results showed that the proposed approach achieves almost 100% of potential switch throughput.

Telephone networks are another area where neural networks have been applied successfully. A notable application is the one proposed to predict traffic fluctuation [69]. Zhang applied a backpropagation neural network to predict traffic fluctuation in circuit-switched telecommunication networks. The simulation results showed that the neural network predicts traffic fluctuation with high precision. Neural network schemes can reduce the prediction error extensively.

Biesterfeld *et al.* [70] used neural networks to solve the problem of *location prediction* in mobile networks. The location prediction means the calculation of the possible future locations of a subscriber based on his historical movement patterns. In the proposed work, a selection of feedforward and feedback neural networks were examined to test their stability. The movement patterns of a subscriber are stored as a function of time and presented to the neural network as a binary coding using a quasi-continuous representation. The results show that the feedforward neural network scheme has its advantages with dynamic motion patterns.

In [71], a neural network was utilized to handle the *alarm correlation in a GSM mobile* telephone network. The GSM mobile network can be divided into three parts: mobile station, network access, and switched network. When a link fails, up to 100 alarms are received by the operation and maintenance center. To avoid overloading the operators, an alarm correlation system is required. In the proposed work, a neural network-based alarm correlator is designed using a feedforward network. Many learning algorithms were compared, such as backpropagation, modified backpropagation, Quickprop, and a special learning algorithm called the Cascade Correlation [72]. The results show that the cascade correlation alarm correlator is well suited for alarm correlation tasks.

Hoang [73] utilized the Hopfield neural network for *communication network topological design*. The aim of the network topological design is to minimize the cost of the network access configuration, which can accommodate traffic demands from many users. The topological design of a computer network is a complex problem that can be broken down into backbone design problem and local access design problem. A particular part of the latter is called the capacitated concentrator location problem (CCLP). CCLP is an optimization problem that minimizes an energy function to estimate the final cost. Hoang's results demonstrated the ability of the Hopfield neural network in solving the optimization of CCLP. The quality of the solutions depends on the setting of the energy function parameters, step size, and the initial values of neurons.

In the study of the applicability of available cluster analysis methods to *multivariate satellite data*, Waldemark [74] studied self-organizing maps (SOM) and Adaptive Resonance Theory (ART) neural networks along with the principal Component Analysis (PCA) and the K-means methods. The results showed that a combined adaptive dynamic K-means (SOM a-dK) procedure achieved useful partition of multivariate satellite data, which had not been studied previously. The generalization capability of SOM a-dK is substantial. It is also reported that the results deduced by SOM a-dK are stable, i.e., independent of who performs the analysis.

### 5.6.6 Neural networks in decision fusion and pattern recognition

Neural networks have been used extensively for pattern recognition and for systems classification. Different approaches and techniques were used to tackle the different aspects of the domains. Recently interest in combining multiple classifiers has been addressed in the literature and a number of methodologies have been proposed ([75], [76], [77], [78], and [79]).

Modular neural networks (MNN) are a new and growing trend in the design of neural-network-based classification systems. Several approaches have been proposed. These approaches include different architecture, learning procedure, and decision aggregation procedures [80]. The basic idea of MNN is to reduce the large size/high complexity classification task into a set of smaller sub-tasks that are easily manageable. The sub-solutions are

integrated via a decision-making strategy. MNN classifiers, generally, proved to be more efficient than the non-modular alternatives. Ensemble networks are one of the most widely used structures of MNN ([81] and [82]). They are basically neural networks having the same structure, yet with different initializations, that are applied to the same classification problem. The redundancy inherent in this scheme acts to the benefit of the overall system. Different approaches were presented in the literature addressing the issue of combining decisions. Huang *et al.* [83] used a neural network to aggregate the different decisions. Jacobs *et al.* [84] took a slightly different approach in which a gating network was used to select the winning expert. Drucker *et al.* ([85] and [86]), and Freund [87], proposed boosting techniques to improve the performance of neural networks.

Cooperative modular neural networks (CMNN) are also an interesting line of modular design [88]. In this case the modules are decoupled into sub-networks that not only learn to accomplish their own task, but try collectively to determine the best module that can provide the correct answer. Generally, MNN proved to be more efficient than their non-modular counterparts.

In any MNN architecture the individual decisions at the module level have to be integrated together using a voting scheme [89]. The individual modules are modeled as multiple voters electing one candidate in a single ballot election, assuming the availability of voters' preference. Some of the voting schemes presented in the literature are: plurality voting, which is by far the most popular, Nash voting, and fuzzy voting ([90] and [79]). All these voting schemes start from the outputs of the individual modules to produce an output without inferring explicit information from the problem feature space. This makes the choice of the aggregation procedure very subjective. Auda and Kamel [80] presented an architecture that integrates the voting scheme in the learning process to help improve the overall performance of the system. In this case the voting scheme is still static and doesn't adapt with the learning process.

## 5.7 Summary

In this chapter we have briefly described topologies and learning algorithms for major classes of neural networks. These include the feedforward and the recurrent-based topologies. They also include networks with supervised and unsupervised learning algorithms. Applications of these networks to different types of real-world problems are also outlined along with their limitations. A lot of work has been carried out in recent years to overcome the main limitations of these connectionist models. A case in point is the integration of tools of computational intelligence and optimization tools to tackle convergence-related problems of the learning algorithms. Some other work has also been carried out for purposes of increasing the storage capacity of recurrent neural networks without necessarily increasing dramatically the number of processing units. The chapter ends with an overview of recent applications of artificial neural networks in industry and other areas. A good number of

references have been included as well to assist readers in assessing what has been done by others in the field. This should serve as a good knowledge base for researchers and scientists working in the field.

## Problems

- Outline the different steps of the backpropagation algorithm for a multilayer perceptron with two inputs, two hidden layers each with two neurons, and one output layer with one output. Presume here that the activation function of the first layer and the second layer are of the logistic type with parameters  $\lambda = 0.5$  and  $\lambda = 1$ , respectively.
- Derive the backpropagation training algorithm for the structure of the feedforward neural network given in problem 1, in which the logistic function is replaced by the hyperbolic tangent function  $\tanh(\alpha x)$  given as:

$$f(x) = \tanh(\alpha x) = \frac{e^{\alpha x} - e^{-\alpha x}}{e^{\alpha x} + e^{-\alpha x}}.$$

Notice that the derivative of  $f(x)$  is given by:

$$\frac{\partial f}{\partial x} = \alpha \operatorname{sech}^2(\alpha x).$$

- This problem illustrates the applicability of MLPs for linearly nonseparable classes. We need to design an MLP with the following structure (parameters): a two-neuron input layer, a two-neuron *logsigmoid* hidden layer, and a one-neuron *linear* output layer.
  - Train the network with the following patterns and once convergence is reached, express the weights vectors among the nodes and the bias terms for each of the nodes:
 

*Class 1:* Input:  $\{[1, 1]^T, [0, -1]^T\}$  Output:  $\{0.2\}$

*Class 2:* Input:  $\{[-1, -1]^T, [1, -1]^T\}$  Output:  $\{-0.2\}$
  - Two more patterns are added to the system,
 

*Class 1:* Input:  $\{[2, 0]^T\}$  Output:  $\{0.2\}$

*Class 2:* Input:  $\{[2, 2]^T\}$  Output:  $\{-0.2\}$

Retrain the network and check whether it converges with the same architecture as before. If not, add a third neuron to the hidden layer and notice the difference in the convergence behavior. What can we conclude here?

- Using a feedforward backpropagation neural network that contains a single hidden layer (with variable number of hidden nodes each having an activation function of the logistic form), investigate the outcome of the neural network for the following mappings:

$$f(x) = \exp(-x^2) \quad x \in [0, 2]$$

$$g(x) = \frac{\pi}{x + \pi} \arctan(x) \quad x \in [0, 1]$$

For each function, create two sets of *input/output* data, one for training and the other for testing.

- (a) Investigate the effect of the number of training patterns (samples) on the accuracy of the output when the number of hidden nodes is fixed (e.g., three hidden nodes).
- (b) Investigate the effect of the number of hidden nodes on the accuracy of the output for a given fixed number of training samples (e.g., 50 training patterns).
- (c) Make qualitative and quantitative deductions in light of these simulations.

5. Repeat the same as in problem 4 for the following mappings:

$$f(x) = \frac{x}{\pi} \tan(x) \quad x \in \left[ \frac{-\pi}{4}, \frac{\pi}{4} \right]$$

$$f(x) = \cos(x) \sqrt[3]{\sin(x)} \quad x \in \left[ 0, \frac{\pi}{2} \right]$$

6. We need to train an MLP network for obtaining the output of the following two-to-one mapping function:

$$y(x_1, x_2) = \sin(2\pi x_1) \cos(0.5\pi x_2)$$

- (a) Set up two sets of data, each of which consists of 100 input–output patterns, one for network training and the other for testing. The input–output data are obtained by randomly varying the input variables ( $x_1, x_2$ ) within the interval  $[-1, 1] \times [-1, 1]$ .
  - (b) First, fix the number of hidden neurons to two (equal to the number of input nodes) and analyze the performance of the obtained network.
  - (c) Analyze the performance of the network with more and then with fewer hidden nodes.
7. Consider a network with  $L$  feedforward layers,  $l = 1, 2, \dots, L$ , and let ' $\text{tot}_i$ ' and ' $\text{o}_i$ ' denote the net input and output of the  $i$ -th unit in the  $l$ -th layer respectively. The network has  $m$  input nodes and  $n$  output nodes. Let ' $w_{ij}$ ' denote the connection weight from ' $\text{tot}_{i-1}$ ' to ' $\text{o}_i$ '.
- (a) Write down in detail the different steps for the backpropagation learning algorithm (seven in total).
  - (b) It is required to train the network below (next page) using the backpropagation learning algorithm outlined above. Each neuron (other than the bias neurons (2), (5), (8)) has an activation function given by function:

$$\text{o}(\text{tot}) = \frac{2}{\pi} \tan^{-1}(\text{tot})$$

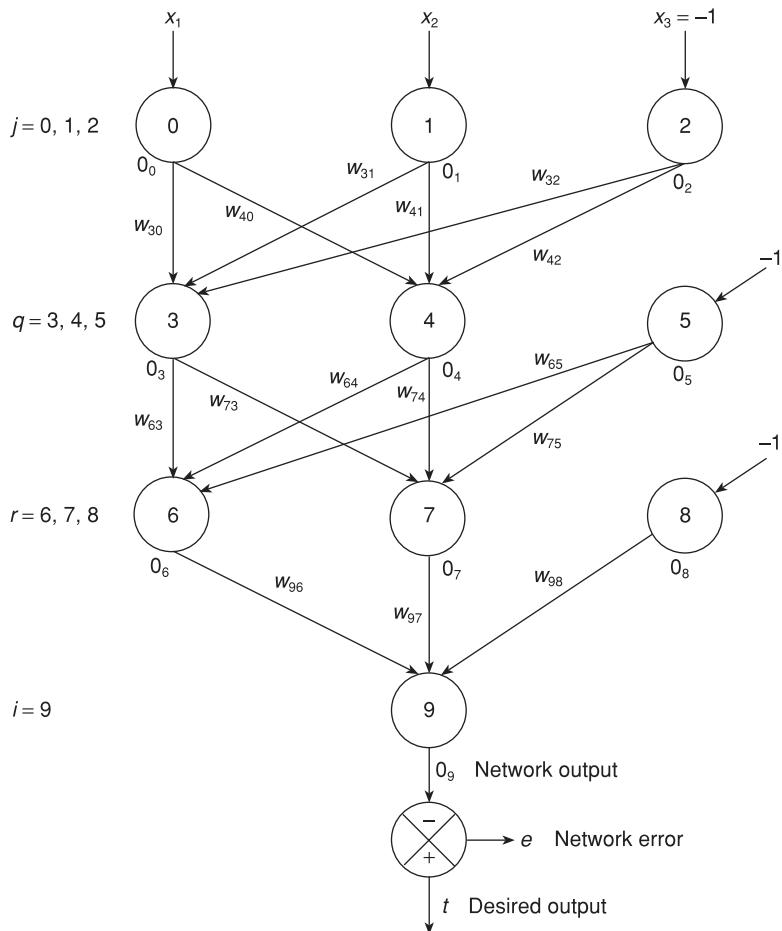
with its derivative given by:

$$\frac{d\text{o}(\text{tot})}{d\text{tot}} = \frac{2}{\pi} \left[ \frac{1}{1 + \text{tot}^2} \right]$$

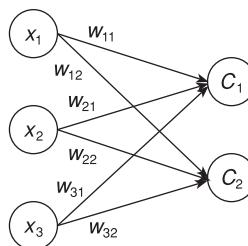
Initialize the weights to small random values.

Set  $\eta = 0.4$  (learning rate),  $E_{\max} = 0.1$  (maximum tolerable error),  $E = 0$  (current error value),  $k = 1$  (current training pattern).

Use one epoch of training and check whether the maximum tolerance error ( $E_{\max}$ ) has been achieved for the training pair  $(\vec{X}, t)$  of data given by  $\vec{X} = (x_1, x_2)^T = (0.3, 0.4)^T$ ,  $t = 0.88$ .



8. Repeat part (a) of problem 6 with two RBF networks: one with Gaussian MFs and the other with exponential MFs. Compare the outcome of both networks. Compare the performance of the networks with that of problem 6 in terms of execution time and accuracy (mean squared errors).
9. Consider a (self-organizing map) network that has three inputs (the input layer) connected to two output neurons (the Kohonen layer) as shown in the diagram below. The connections form two weight vectors:  $\mathbf{w}_1$  and  $\mathbf{w}_2$ .



Given the initial weight vectors:

$$\mathbf{w}_1 = (0, 1, 1); \quad \mathbf{w}_2 = (1, 1, 0)$$

and the training examples:

$$\mathbf{x}_1 = (0, 0, 1); \quad \mathbf{x}_2 = (0, 1, 0); \quad \mathbf{x}_3 = (1, 0, 0)$$

Calculate the updated weight vectors resulting from training with each successive training example and enter them in a table similar to the one shown below. Note that this means that the changes made after the first example apply to training with the second example, etc. Use a constant learning rate,  $\alpha = 1.0$ , and assume a neighborhood of zero (only one weight vector is updated).

$w_1$	$w_2$
After training with $\mathbf{x}_1$	
After training with $\mathbf{x}_2$	
After training with $\mathbf{x}_3$	

10. We need to store the fundamental memory given by the vector  $V = [1 \ 1 \ -1 \ 1]$  in a four-node Hopfield network with zero threshold values. Using the different steps of training, deduce the structure of the network, extract the energy levels for each possible state introduced to the network and display the transition table and the transition diagram.
11. Repeat problem 10, while choosing as threshold the values of 0.2, 0.4, 0.2, 0.7. What are the main differences from the result obtained in problem 8?
12. Repeat problem 10 while storing the patterns given by the vectors  $V_1 = [1 \ 1 \ -1 \ 1]$  and  $V_2 = [1 \ -1 \ 1 \ 1]$ .

## References

1. Werbos, P. (1974) "Beyond regression: new tools for prediction and analysis in the behavioral sciences," PhD dissertation, Harvard University.
2. Cybenko, G. (1989) "Approximation by superposition of a sigmoidal function," *Mathematics of Controls, Signals and Systems*, 2: 303–14.
3. Rumelhart, D., Hinton, G., and Williams, R. (1986) "Learning representations by backpropagation errors," *Nature*, 323: 533–6.
4. Haykin, S. (1994) *Neural Networks, a Comprehensive Foundation*, MacMillan Publishing, Englewood Cliffs, NJ.
5. Jang, J., Sun, S., and Mizutani, E. (1997) *Neuro-Fuzzy and Soft Computing*, Prentice Hall, Upper Saddle River, NJ.
6. Shar, S., and Palmieri, F. (1990) "MEKA, a fast local algorithm for training feed-forward neural networks," in *Proceedings of the International Joint Conference on Neural Networks*, 3: 41–6.
7. Powell, M. (1985) "Radial basis function for multivariable interpolation: a review," in *Proceedings of IMA Conference on Algorithm for the Approximation of Function and Data*, RMCS, Shrivnham.
8. Katayama, R., Kuwata, K., Kajitani, Y., and Watanabe, M. (1995) "Embedding dimension estimation of chaotic time series using self-generating radial basis function network," *Fuzzy Sets and Systems*, 72(3): 311–27.

9. Hopfield, J. (1984) "Neurons with graded response have collective computational properties like those of two state neurons," in *Proceedings of the National Academy of Science*, pp. 3088–92.
10. Hopfield, J., and Tank, D. (1985) "Neural computation of decisions in optimization problems," *Biological Cybernetics*, 52: 141–52.
11. Huetter, G. (1988) "Solution of the traveling salesman problem with an adaptive ring," in *Proceedings of the International Conference on Neural Networks*, pp. 85–92.

[References for Neural Net in Process Monitoring and Optimal Control and for Neural Networks in Semiconductor Manufacturing Processes]

12. Holland, J.H. (1975) *Adaption in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor.
13. Holland, J.H., Hoyer, K.J., Nisbett, R.E., and Thagard, P.R. (1986) *Induction: Processes of Inference, Learning, and Discovery*, MIT Press, Cambridge, Mass.
14. Goldberg, D. (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, Mass.
15. Fogel, L.J., Owens, A.J., and Walsh, M.J. (1966) *Artificial Intelligence Through Simulated Evolution*, J. Wiley & Sons, New York.
16. Aarts, E. (1989) *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to combinatorial Optimization and Neural Computing*, J. Wiley & Sons, New York.
17. Yip, P.P.C., and Pao, Y.H. (1994) "A guided evolutionary simulated annealing approach to the quadratic assignment problem", *IEEE Transactions on Systems, Man and Cybernetics*, vol. 24, no. 9, pp. 1385–8, September 1994.
18. Pao, Y.H., and Yip, P.P.C. (1996) *Fuzzy Logic and Neural Network Handbook*, McGraw Hill, New York.
19. May, G. (1996) *Fuzzy Logic and Neural Network Handbook*, McGraw Hill, New York.
20. Huang, S., and Zhang, H. (1994) "Artificial neural networks in manufacturing: concepts, applications and perspectives," *IEEE Transactions on Component Packaging and Manufacturing Technology*, Part A, vol. 17, no. 2, June 1994.
21. Hopfield, J., and Tank, D. (1985) "Neural computation of decisions in optimization problems," *Biological Cybernetics*, vol. 52.
22. Lippman, R. (1987) "An introduction to computing with neural nets," *IEEE ASSP Magazine*, April 1987.
23. Himmel, C., and May, G. (1993) "Advantages of plasma etch modeling using neural networks over statistical techniques," *IEEE Transactions on Semiconductor Manufacturing*, vol. 6, no. 2, May 1993.
24. Mocella, M., Bondur, J., and Turner, T. (1991) "Etch process characterization using neural network methodology: a case study," *SPIE Proceedings on Process Module Metrology, Control and Clustering*, vol. 1594.
25. Owens, A., and Mocella, M. (1991) "An experimental design advisor and neural network analysis package," *Proceedings of the IEEE International Workshop on Artificial Neural Networks*, September.
26. Rietman, E., and Lory, E. (1993) "Use of neural networks in semiconductor manufacturing processes: an example for plasma etch modeling," *IEEE Transactions on Semiconductor Manufacturing*, vol. 6, no. 4, November 1993.
27. Kim, B., and May, G. (1994) "Modeling reactive ion etching of silicon dioxide films using neural networks," *Proceedings of the 1994 Electronic Components and Technology Conference*, May.

28. Nami, Z., Erbil, A., and May, G. (1994) "Semi-empirical MOCVD modeling using neural networks," *Proceedings of the 1994 SPIE Conference on Microelectronics Manufacturing*, October 1994.
29. Nadi, F., Agogino, A., and Hodges, D. (1991) "Use of influence diagrams and neural networks in modeling semiconductor manufacturing processes," *IEEE Transactions on Semiconductor Manufacturing*, vol. 4, no. 1, February 1991.
30. Han, S., and May, G. (1994) "Modeling with plasma enhanced chemical vapor deposition process using neural networks and genetic algorithms," *Proceedings of the 6th IEEE International Conference on AI Tools*, New Orleans, November.
31. Baker, M., Himmel, C., and May, G. (1995) "Time series modeling of reactive ion etching using neural networks," *IEEE Transactions on Semiconductor Manufacturing*, vol. 8, no. 1, February 1995.
32. Yazici, H., and Smith, A. (1993) "Neural network control charts for location and variance process shifts," *Proceedings of the 1993 World Congress on Neural Networks*, vol. I.
33. Nelson, D., Ensley, D., and Rogers, S. (1992) "Prediction of chaotic time series using cascade correlation: effects of number of inputs and training set size," *SPIE Conference on Applications of Neural Networks*, vol. 1709.
34. Mori, H., and Ogasawara, T. (1993) "A recurrent neural network approach to short-term load forecasting in electrical power systems," *Proceedings of the 1993 World Congress on Neural Networks*, vol. 1.
35. Rao, S., and Pappu, R. (1993) "Nonlinear time series prediction using wavelet networks," *Proceedings of the 1993 World Congress on Neural Networks*, vol. IV.
36. Anderson, K., Cook, G., and Gabor, K. (1990) "Artificial neural networks applied to arc welding process modeling and control," *IEEE Transactions on Industrial Applications*, vol. 26.
37. Hattori, S., Nakajima, M., and Katayama, Y. (1992) "Fuzzy control algorithms and neural networks for flatness control of a cold rolling process," *Hitachi Review*, vol. 41, no. 1.
38. Lam, M., Lin, P., and Bain, L. (1992) "Modeling and control of the lithographic offset color printing process using artificial neural networks," *Neural Networks in Manufacturing and Robotics*, ASME, vol. 57.
39. May, G. (1994) "Neural networks aid IC manufacturing," *IEEE Spectrum*, September.
40. Sorsa, T., Koivo, H., and Koivisto, H. (1991) "Neural networks in process fault diagnosis," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 21, no. 5.

#### References for Power

41. Weerasooriya, S., and El-Sharkawi, M. (1991) "Use of Karhunen-Loe've expansion in training neural networks for static security assessment," *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, IEEE, New York, NY, USA, pp. 59–64.
42. Hsu, Y., and Chen, C. (1991) "Tuning of power system stabilizers using an artificial neural network," *IEEE Transactions on Energy Conversion*, vol. 6, no. 4, December 1991, pp. 612–9.
43. Zhang, Y., Chen, G.P., Malik, O.P., and Hope, G.S. (1993) "An artificial neural network-based adaptive power system stabilizer," *IEEE Transactions on Energy Conversion*, vol. 8, no. 1, March 1993, pp. 71–9.
44. Farag, W., Quintana, V., and Lambert, T. (1998) "Enhancing the transient stability of multi-machine power systems using intelligent techniques," *Bulk*

- Power Systems Dynamics and Control – IV Restructuring. Symposium Proceedings*, Athens, Greece, pp. 117–25.
45. Kraft, T., Ogakagi, K., Ishii, R., and Surko, P. (1991) “A hybrid neural network and expert system for monitoring fossil fuel power plants,” *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, pp. 215–8.
  46. Amjadi, N., and Ehsan, M. (1999) “Evaluation of power systems reliability by an artificial neural network,” *IEEE Transactions on Power Systems*, vol. 14, no. 1, February 1999, pp. 287–92.
  47. El-Sayed, M. (1998) “Artificial neural network for reactive power optimization,” *Neurocomputing*, vol. 23, nos 1–3, December 1998, pp. 255–63.
  48. Srinivasan, D., Tan, S., Chang, C., and Chan-E, E. (1998) “Practical implementation of a hybrid fuzzy neural network for one-day-ahead load forecasting,” *IEEE-Proceedings Generation, Transmission and Distribution*, vol. 145, no. 6, November 1998, pp. 687–92.
  49. Wu, Q., Hogg, B., and Irwin, G. (1991) “On line training of neural network model and controller for turbogenerators,” *Proceedings of the First International Forum on Applications of Neural Networks to Power Systems*, pp. 161–5.

### References for Robotics

50. Cui, X., and Shin, K.G. (1996) “Intelligent coordination of multiple systems with neural networks,” in *Intelligent Control Systems: Theory and Applications*, M.M. Gupta and N.K. Sinha, eds, pp. 206–33, IEEE Press.
51. Shibata, T., Fukuda, T., Shiotani, S., Mitsuoka, T., and Tokita, M. (1993) “Hierarchical hybrid neuromorphic control system for robotic manipulators,” *JSME International Journal*, Series C, vol. 36, no. 1, pp. 100–9.
52. Shibata, T., and Fukuda, T. (1994) “Hierarchical intelligent control for robotic motion,” *IEEE Transactions on Neural Networks*, vol. 5, no. 5, pp. 823–32, September 1994.
53. Shi, X., Lever, P.J.A., and Wang, F.Y. (1996) “Experimental robotic excavation with fuzzy logic and neural networks,” *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, Minneapolis, Minnesota, pp. 957–62, April 1996.
54. Zheng, X.Z., and Ito, K. (1997) “Self-organized learning and its implementation of robot movements,” *IEEE International Conference on Systems, Man, and Cybernetics*, vol. 1, pp. 281–6.
55. Ritter, H.J., Martinez, T.M., and Schulten, K.J. (1989) “Topology-conserving maps for learning visuomotor coordination,” *Neural Networks*, vol. 2, pp. 159–68.
56. Martinez, T.M., Ritter, H.J., and Schulten, K.J. (1990) “Three-dimensional neural net for learning visuomotor coordination of a robot arm,” *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 131–6.
57. Walter, J.A., Martinez, T.M., and Schulten, K.J. (1991) “Industrial robot learns visuomotor coordination by means of ‘neural-gas’ network,” *Proceedings of the International Conference on Artificial Neural Networks*. T. Kohonen *et al.*, eds, vol. 1, pp. 357–64, Elsevier Science, North-Holland.
58. Walter, J.A., and Schulten, K.J. (1993) “Implementation of self-organizing neural networks for visuomotor control of an industrial robot,” *IEEE Transactions on Neural Networks*, vol. 4, no. 1, pp. 86–95.
59. Luo, Z.W., Asada, K., Yamakita, M., and Ito, K. (1994) “Self-organization of a uniformly distributed visuomotor map through controlling the spatial variation,”

- Distributed Autonomous Robotic Systems*, H. Asama *et al.*, eds, pp. 279–88, Springer-Verlag, Tokyo.
60. Fierro, R., and Lewis, F.L. (1998) “Control of a nonholonomic mobile robot using neural networks,” *IEEE Transactions on Neural Networks*, vol. 9, no. 4, pp. 589–600, July 1998.
  61. Fierro, R., and Lewis, F.L. (1997) “Robust practical point stabilization of a nonholonomic mobile robot using neural networks,” *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 20, nos. 2–4, pp. 295–317, October–December 1997.
  62. Torras, C., Cembrano, G., Millan, J. del R., and Wells, G. (1995) “Neural approaches to robot control: four representative applications,” *Proceedings of the International Workshop on Artificial Neural Networks*, pp. 1016–35, Springer-Verlag, Berlin, Germany.
  63. Meng, R., and Picton, P.D. (1994) “Neural network for local guidance of mobile robots,” *Proceedings of the Third International Conference on Automation, Robotics and Computer Vision*, vol. 2, pp. 1238–42, Nanyang Technol. Univ, Singapore.
  64. Inigo, R.M., and Torres, R.E. (1995) “Mobile robot navigation with vision based neural networks,” *Proceedings of the SPIE, The International Society for Optical Engineering*, vol. 2352, pp. 68–79.
  65. Ito, H., Furuya, T., and Hagiwara, T. (1995) “Intelligent mobile robot,” *Proceedings of the IEEE International Conference on Neural Networks*, vol. 5, pp. 2699–702, New York, USA.

#### References for Communications

66. Yousef, S., Habib, I., and Saadawi, T. (1997) “A neurocomputing controller for band-width allocation in ATM networks,” *IEEE Journal Selected Areas in Communications*, Feb. 1997.
67. Tarraf, A., Habib, I., and Saadawi, T. (1994) “A novel neural network controller using reinforcement learning method for ATM traffic policing,” *IEEE Journal Selected Areas in Communications*, August 1994.
68. Park, Y., Cherkassky, V., and Lee, G. (1994) “Omega network-based ATM switch with neural network-controlled bypass queueing and multiplexing,” *IEEE Journal Selected Areas in Communications*, vol. 12, no. 9, pp. 1471–80, December 1994.
69. Yongzheng-Zhang (1996) “Prediction of traffic fluctuation in telephone networks with neural networks,” *ICCT '96, International Conference on Communication Technology Proceedings*. IEEE, New York, USA, vol. 2, pp. 909–12.
70. Biesterfels, J., Ennigrou, E., and Jobmann, K. (1997) “Neural networks for location prediction in mobile networks,” *Proceedings of the Int. Workshop on Applications of Neural Networks to Telecommunications*, pp. 140–8, June 1997.
71. Wietgrefe, H., Tuchs, K., Jobmann, K., Carls, G., Frohlich, P., Nejdl, W., and Steinfels, S. (1997) “Using neural networks for alarm correlation in cellular phone networks,” *Proceedings of the Int. Workshop on Applications of Neural Networks to Telecommunications*, pp. 248–55, June 1997.
72. Fahlman, S., and Lebiere, Chr. (1990) “The cascade-correlation learning architecture,” in *Advances in Neural Information Processing Systems 2*, D. Touretzley (ed.), Morgan Kaufmann, San Mateo, CA, pp. 524–32.
73. Hoang, D. (1997) “Neural networks for network topological design,” *Proceedings of the Int. Workshop on Applications of Neural Networks to Telecommunications*, pp. 207–14, June 1997.

74. Waldemark, J. (1997) "An automated procedure for cluster analysis of multi-variate satellite data," *International Journal of Neural Systems*, vol. 8, no. 1, February 1997.

### References for Decision Fusion and Pattern Recognition

75. Kittler, J., Hatef, M., Duin, R., and Matas, J. (1998) "On combining classifiers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 3, pp. 226–39.
76. Auda, G., and Kamel, M. (1998) "Modular neural network classifiers: a comparative study," *Journal of Intelligent and Robotic Systems*, vol. 21, pp. 117–29.
77. Drucker, H., Schapire, R., and Simard, P. (1993) "Improving performance in neural networks using a boosting algorithm," *Neural Information Processing Systems*, vol. 5, pp. 42–9.
78. Cho, S.-B., and Kim, J. (1995) "Combining multiple neural networks by fuzzy integral for robust classification," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 25, no. 2, pp. 380–4.
79. Battiti, R., and Colla, A.M. (1994) "Democracy in neural nets: voting schemes for classification," *Neural Networks*, vol., no. 4, pp. 691–707.
80. Auda, G., and Kamel, M. (1998) "EVOL: ensemble voting on-line", *International Conference on Neural Networks*, pp. 1356–60.
81. Ahn, J., Kim, J., and Cho, S. (1995) "Ensemble competitive learning neural networks with reduced input dimensions," *International Journal of Neural Systems*, vol. 6, no. 2, pp. 132–42.
82. Krogh, A., and Vedelsby, J. (1994) "Neural network ensembles, cross validation and active learning," In *Neural Information Processing Systems, Natural and Synthetic*, pp. 231–8.
83. Huang, Y., Liu, K., and Suen, C. (1995) "The combination of multiple classifiers by a neural network approach," *Journal of Pattern Recognition and Artificial Intelligence*, vol. 9, pp. 579–97.
84. Jacobs, R.A., Jordan, M.I., Nowlan, S.J., and Hinton, G.E. (1991) "Adaptive mixtures of local experts," *Neural Computation*, vol. 3, pp. 79–87.
85. Drucker, H., Schapire, R., and Simard, P. (1993) "Boosting performance in neural networks," *Journal of Pattern Recognition and Artificial Intelligence*, vol. 7, no. 4, pp. 704–9.
86. Drucker, H., Cortes, C., Jackel, L.D., LeCun, Y., and Vapnik, V. (1994) "Boosting and other ensemble methods," *Neural Computation*, vol. 6, pp. 1289–1301.
87. Freund, Y. (1990) "Boosting a weak learning theory," *Proceedings of the Third Annual Workshop Computational learning theory*, Rochester, NY, pp. 202–6.
88. Auda, G., and Kamel, M. (1997) "CMNN: cooperative modular neural networks for pattern recognition," *Pattern Recognition Letters*, vol. 18, pp. 1391–8.
89. Auda, G., Kamel, M., and Raafat, H. (1995) "Voting schemes for cooperative neural network classifiers," *International Conference on Neural Networks*, Australia, vol. 3, pp. 1240–3.
90. Rogova, G. (1994) "Combining the results of several neural network classifiers," *Neural Networks*, vol. 7, no. 5, pp. 777–81.

# Dynamic neural networks and their applications to control and chaos prediction

## Chapter outline

6.1	Introduction	299
6.2	Background	300
6.3	Training algorithms	304
6.4	Fields of applications of RNN	306
6.5	Dynamic neural networks for identification and control	307
6.6	Neural network-based control approaches	313
6.7	Dynamic neural networks for chaos time series prediction	323
6.8	Summary	330
	Problems	330
	References	332

## 6.1 Introduction

As mentioned previously, neural network structures can be generally classified into feedforward or recurrent architecture. Feedforward networks are networks in which the output of a given node is always injected into the following layer. Recurrent networks, on the other hand, have some outputs directed back as inputs to node(s) in the same or preceding layer. The presence of cycles in a recurrent network provides it with the abilities of dynamically encoding, storing, and retrieving context information [1], hence their name as *dynamic neural networks*. Much like a sequential circuit, the state of the network at a given instant depends on the state at the previous instant. This capability of handling state information opens the door to many useful

applications, some of which are discussed in this chapter. The basic concept of recurrent networks and their dynamics are reviewed here. Their variety and their training algorithms are also outlined. We discuss their application for identification and control of dynamic processes, and chaotic systems prediction. While possessing several attractive capabilities, recurrent neural networks remain among the most challenging networks to deal with, given the variety of their structures and the sometimes complex and computational intensive training process. Several advances have been made to standardize their use and are mentioned in the references provided at the end of the chapter.

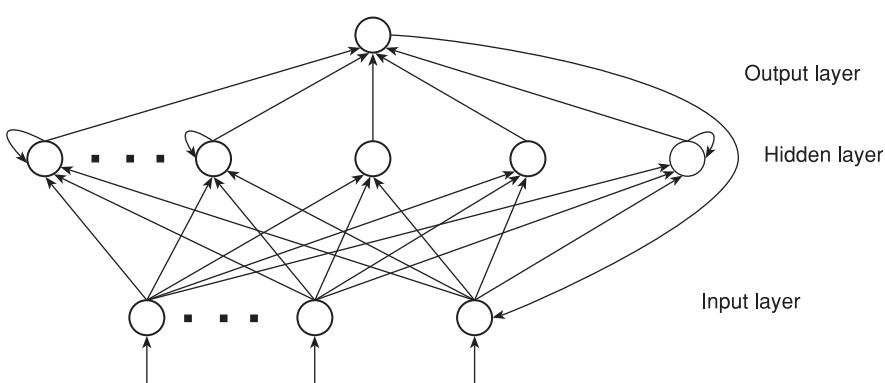
## 6.2 Background

### 6.2.1 Basic concepts of recurrent networks

The architecture is similar to the feedforward network except that there are connections going from nodes in the output layer to the ones in the input layer. There are also some self-loops in the hidden layer. Figure 6.1 shows a typical structure of a recurrent neural network (RNN). The “feedback” connections are specific to these networks and allow them to tackle problems involving dynamic processes and spatiotemporal patterns.

The operation of a recurrent node can be summarized as follows. When time inputs are presented at the input layer, the unit computes the activation output just like a feedforward node does. However, its net input consists of additional information reflecting the state of the network. For the subsequent inputs, the state will essentially be a function of the previous and current inputs. As a result, the behavior of the network depends not only on the current excitation, but also on past history.

The wealth of information that can be stored in a recurrent network can prove to be very useful. At the same time, however, the feedback connections pose a challenging problem in terms of training, as the two basic requirements



**Figure 6.1:** A typical structure of recurrent network

for computing activations and errors are violated [2]. For an arbitrary unit in a recurrent network, the activation at a given time  $t$  is given as

$$o_i(t) = f(\text{tot}_i(t-1)) = f\left(\sum_j w_{ij} o_j(t-1) + x_i(t-1)\right) \quad (6.1)$$

The notation used is similar to what has been adopted earlier for the static BPL, except that we have here a time-dependence for all the signals. In particular,  $o_i(t)$  is the current output signal of node  $i$ ,  $\text{tot}_i(t-1)$  is the net signal input to node  $i$  at time “ $t-1$ ”,  $o_j(t-1)$  is the output signal of the node  $j$  at previous time sample, and  $w_{ij}$  is the connection weight between node  $i$  and node  $j$ . The computation of  $o_i$  requires the knowledge of the activations of all units in the *posterior* set, which consists of nodes whose output was relayed to unit  $i$ . When computing errors, the errors of all units in the *anterior* set of nodes will have to be determined beforehand.

### 6.2.2 The dynamics of recurrent neural networks

Dynamic systems and non-stationary processes abound in nature and in real-world applications. From small servo systems to the largest power and chemical plants, the information of the previous system states and the actual ones are always relevant for making the appropriate decisions in driving the system to its required state, or in predicting its future state.

To appreciate the dynamical property of recurrent networks, consider static feedforward networks that can only map representations from one space to another. The states of all the nodes in the static network are determined once the signals from the input nodes have propagated throughout the network. These types of networks have no nontrivial dynamics and are considered memoryless. Hence they are incapable of tackling temporal tasks-based processes. Recurrent networks, on the other hand, are capable of encoding, storing and processing the current context for later use. Their computational units assume activities based on the history of the network. One of their major characteristics is that they are able to map sequences of input vectors distributed across time into output vector sequences [1].

### 6.2.3 Architecture

The defining feature of recurrent neural networks is their use of feedback signals transmitted throughout the layers of neurons. The feedback signals can be either self-loops or backward connections as illustrated in Figure 6.2.

Recurrent neural networks can be classified into two categories pertaining to their weight connections. The first group has symmetrical weight connections. Hopfield networks are an example of this category and Figure 6.3 illustrates their structure. Introduced by Prof. J. Hopfield in 1982, Hopfield networks have feedback links with symmetrical weights from each output signal to all other output nodes, except the node itself (i.e., no self-loops).

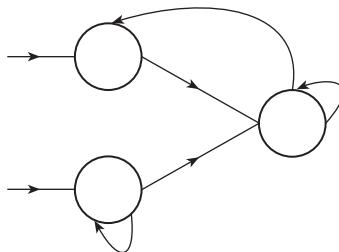


Figure 6.2: Feedback signals among neurons of RNN

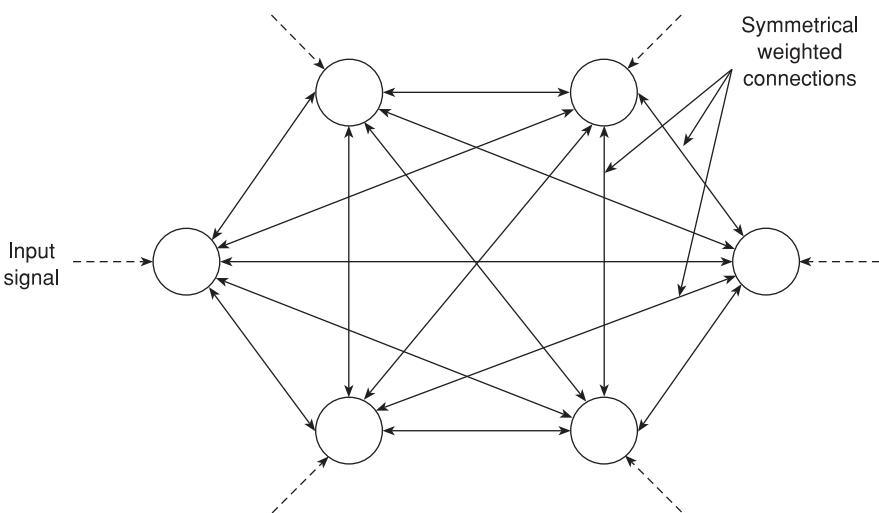


Figure 6.3: Typical representation of the Hopfield neural network

Hopfield networks and their applications have been discussed in detail in an earlier chapter. The second category, which is the focus of this chapter, is characterized by asymmetrical weight connections. The two major types of recurrent networks in this category are *partially recurrent networks* and *fully recurrent networks*.

A partially recurrent network is basically a multilayer feedforward network, but with feedback links coming from either the hidden-layer nodes or the output nodes [3]. The layer to which the feedback links are connected is an artificially added layer called the *context layer*. The two well-known models of this type are the simple recurrent network (SRN; also called the Elman network) and Jordan's sequential network.

The recurrent connections in an SRN are from the hidden layer to the context layer as shown in Figure 6.4. In Jordan's sequential network as shown in Figure 6.5, connections are directed from the output layer to the context layer, with additional self-connections within the context layer.

In contrast, the fully recurrent networks allow for any node of the network to be connected to any other node [4]. This is illustrated in Figure 6.6.

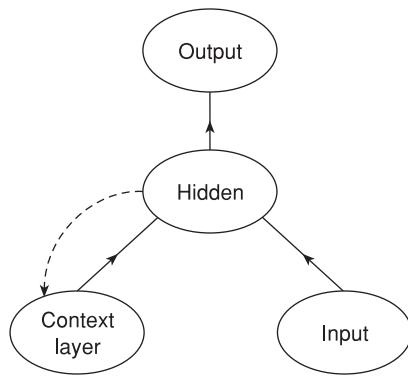


Figure 6.4: Architecture of a simple recurrent network (Elman network)

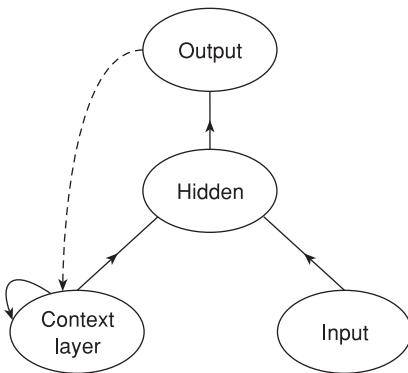


Figure 6.5: Architecture of Jordan's sequential network

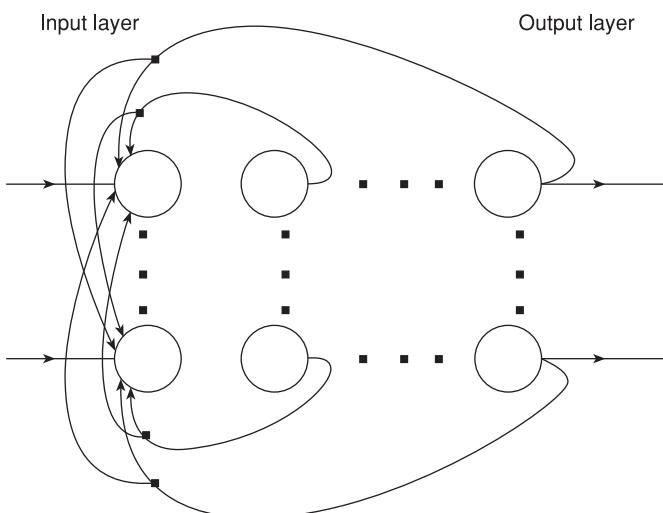


Figure 6.6: Architecture of a fully recurrent network

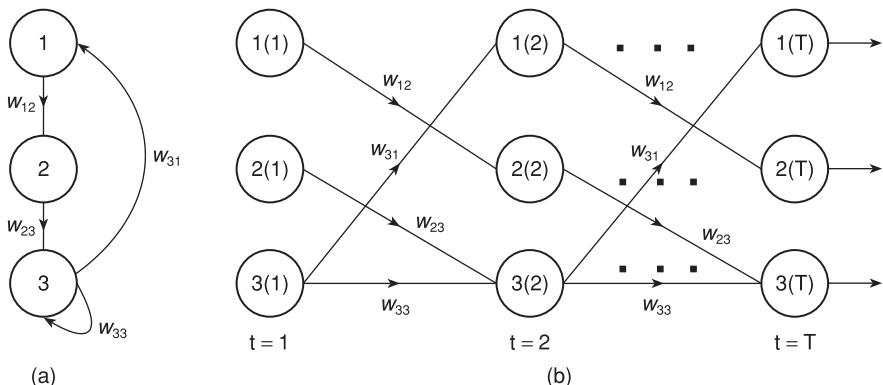
The feedback connections within recurrent networks make them eligible candidates for applications involving complex dynamical, spatiotemporal behavior [1]. The capability of storing and processing *context information* or *input history* allows them to solve problems that cannot be solved by feedforward networks. However, this comes at the expense of more complex learning and often longer training time. This is discussed in the next sections.

## 6.3 Training algorithms

The search for the most effective training algorithm for recurrent networks is still an ongoing research problem. This is particularly challenging due to the architectural diversity of recurrent networks. As a result, numerous training approaches have been proposed and studied in the literature in the last decade and the majority of them have discussed ways to speed up the training process. In this section, a number of these algorithms are reviewed and summarized to give readers a flavor of how recurrent networks can be trained. They often find practical applications in areas such as nonlinear modeling and control, pattern and sequence processing based on their structural and convergence characteristics.

### 6.3.1 Backpropagation through time (BPTT)

The BPTT training method is an extension of the standard backpropagation algorithm and is based on an off line training procedure. The main idea is to transform the recurrent network from a spatio-temporal representation into a standard representation where artificial layers are created simulating the network structure frozen in time for each sequence of the data obtained. In other words, we transform the recurrent network into an equivalent feedforward network by unfolding it into artificial layers. The unfolded network can then be trained using the conventional backpropagation algorithm, in a similar way as training a standard feedforward network with duplicated weights. An illustration of the unfolding of a sample network is given in Figures 6.7(a) and 6.7(b).



**Figure 6.7:** Three-unit recurrent network (a) and its feedforward network equivalent (b)

A generalized version of the BPTT algorithm has been derived in detail in [5] and has been applied to a relatively general set of Layered Digital Dynamic Networks (LDDN).

### 6.3.2 Real-time backpropagation learning

In contrast to the off line weight adaptation of the BPTT representation, the real-time backpropagation learning (RTBL) algorithm deals with an on line weight adaptation scheme. Presume we are dealing with a fully recurrent neural network where each node of the network has an output signal given by

$$o_i(t) = f(\text{tot}_i(t-1)) = f\left(\sum_j w_{ij} o_j(t-1) + x_i(t-1)\right) \quad (6.2)$$

and where the learning occurs during presentation of sequences of signals. If we denote by  $E_{cuml}$  as being the cumulative error over the total spectrum time “T” of all output node errors  $E_l(t)$ :

$$E_{cuml} = \sum_t \frac{1}{2} \sum_l E_l^2(t) \quad (6.3)$$

where  $E_l(t)$  being the error at time  $t$  between target  $d_l$  and output  $o_l$  at the output layer  $l$  with  $E_l(t) = d_l(t) - o_l(t)$ .

As in the BPL, we wish to minimize the cumulative error  $E_{cuml}$  in the weight space:

$$\frac{\partial E_{cuml}}{\partial w_{ij}(t)} = \sum_t \frac{\partial E_{cuml}}{\partial \text{tot}_i(t)} \frac{\partial \text{tot}_i(t)}{\partial w_{ij}(t)}, \quad t = 0, 1, \dots, T \quad (6.4)$$

Using the gradient descent technique, we define the update of the weight as:

$$\Delta w_{ij}(t) = -\eta \frac{\partial E_{cuml}}{\partial w_{ij}} = \eta \sum_l E_l \frac{\partial o_l(t)}{\partial w_{ij}} \quad (6.5)$$

where  $\eta$  is the learning rate parameter and

$$\frac{\partial o_l(t)}{\partial w_{ij}} = f'(\text{tot}_l(t-1)) \left[ \delta_{li} o_j(t-1) + \sum_p w_{lp} \frac{\partial o_p(t-1)}{\partial w_{ij}} \right] \quad (6.6)$$

where  $\delta_{ij} = \begin{cases} 1 & \text{if } i=j \\ 0 & \text{if } i \neq j \end{cases}$

If we denote by  $\frac{\partial o_p(t)}{\partial w_{ij}} = v_{ij}^{(p)}$ , then we have

$$v_{ij}^{(l)} = f'(\text{tot}_l(t-1)) \left( \delta_{li} o_j(t-1) + \sum_p w_{lp} v_{ij}^{(p)}(t-1) \right) \quad (6.7)$$

and

$$\Delta w_{ij}(t) = \eta \sum_l E_l v_{ij}^{(l)}(t)$$

The total weight correction over the whole set of sample times  $t = 0, 1, \dots, T$  is then given as

$$\Delta w_{ij, Total} = \eta \sum_t \sum_l E_l v_{ij}^{(l)}(t), \quad t = 0, 1, \dots, T \quad (6.8)$$

But while this method allows for on line adaptation, the main drawback remains its computational complexity. Because both derivatives and node outputs evolve recursively, the calculation of the derivatives of node outputs with respect to the weights must be performed during the forward propagation of the signals [1]. In asymptotic notation,  $O(n^4)$  computations are performed for each data point, where  $n$  indicates the number of nodes in the network.

A form of *truncated backpropagation through time* offers potential computational speedups and relatively less storage requirement. This method can be executed at every time step with a suitable truncation depth  $h$ . The derivatives obtained closely approximate those of the real-time method, but with reduced complexity and less computational effort [1].

In the consideration of the computation complexity, the RTBL can be modified by replacing the actual output  $o_l(t)$  with the desired output signal  $d_l(t)$  in the subsequent computing of the target network. The error  $E_l(t)$  and the set of derivatives should then be derived in advance. This technique is called *teacher forcing* and may allow to a great extent the network following the desired data set while accelerating the learning process [3].

The gradient approaches, with all their limitations, have been very often used as standard training methods. This is due to the solid mathematical rationale for using such approaches. However, the problems of high computational complexity and slow convergence remain the main disadvantages of such approaches. In [6], a novel formulation and a unified algorithm based on several conventional approaches have been proposed. Instead of computing the exact gradient, this approach obtains a good approximation of the gradient, which can then be efficiently computed [6].

## 6.4 Fields of applications of RNN

In the past two decades, the applications of recurrent networks in various fields have been documented extensively in the literature. The variety of applications using RNNs is large. Various architectures and training techniques have

been used in a wide spectrum of applications ranging from portfolio management to modeling linguistic behaviors and from modeling of large chemical plants to the design of micro motors [1]. For instance, Wang *et al.* used a simple three-layer RNN and the BPTT algorithm to perform speech-to-text conversion [7]. Over 70% of character recognition accuracy was achieved. Kermanshahi applied a combination of an RNN and an FFBP for long-term power system load forecasting [8]. Twenty years of field data have been used for training and a sliding window training method has been used for continuous adaptation. The application of RNNs in financial forecasting has been often considered with skepticism by researchers and experts in the field, but there is ongoing research in this area as well. In the area of computer science, RNNs have been used to process arbitrarily shaped data structures to encode complex relationships between basic entities [1]. Other examples include, but are not restricted to, structural and syntactic pattern recognition, computational biology, and theorem proving. But still, some of the major applications of recurrent neural networks remain in the area of control of nonlinear systems and chaotic system prediction. In the remaining portion of this chapter we outline the fundamentals of these specific applications and compare them with other techniques.

## 6.5 Dynamic neural networks for identification and control

### 6.5.1 Background

Model-based control techniques are usually implemented under the assumption of a good understanding of the process dynamics and its operating environment. These techniques, however, may not provide satisfactory results when applied to poorly modeled processes, which may operate in ill-defined environments. Even when a suitable analytical model is available, the model parameters might not always be completely known. This is often the case when dealing with complex dynamical systems for which the physical processes are not fully understood. This ultimately leads to difficulties in deriving an adequate model for the system. For processes for which experimental input-output data are available, system identification techniques have been shown to be quite effective in deriving a good dynamic representation for the system. This is particularly useful for designing adaptive control algorithms. For accomplishing this, a model is first derived using identification schemes. The identified model is then used as a starting point for designing a controller with adjustable gains, making the system capable of sustaining relatively large variations in the system parameters, while providing satisfactory performance. Adaptive control schemes have been generally applied for plants with partially unknown dynamics and/or for plants with slowly time-varying parameters.

In response to the growing demand for designing sophisticated and highly reliable processes, designers from many fields have developed systems

characterized by a high level of component integration and quality precision. This, however, has come at the expense of a substantial increase in systems complexities, and has created a new class of challenging problems in terms of modeling and control aspects. Such complex processes are usually characterized by highly nonlinear and time-varying behavior, extensive dynamic coupling, hierarchies, multiple time scales, and high dimensional decision space. As a result, their control may no longer be satisfied by the existing control theory including adaptive control and other known robust control techniques such as  $H$ -infinity ( $H_\infty$ ) and linear-quadratic-Gaussian (LQG) based methods [9–10]. Conventional systems identification tools have their own limitations too, as they may not be able to represent faithfully the dynamics of the process. This happens when, for instance, the controlled variables are not fully accessible or when the system is highly nonlinear, so that describing it with a linear model or series of linear models would not be satisfactory. Figure 6.8 outlines various categories of dynamic systems and the classes of controllers used to deal with them.

Research carried out in the area of knowledge-based systems (KBS) and connectionist modeling (CM) as applied to process and system control has shown that it is possible to circumvent some of the aforementioned difficulties by designing a new family of systems, which some have termed *intelligent controllers*. One major feature of these systems is their capabilities for learning and autonomous action. This makes them highly adaptable even to significant and unanticipated changes of the process dynamics and its environment. Among the techniques that have been proposed in this respect are those based on neural networks, which when implemented within the control loop can serve as good model identifiers and controllers. Due to their proven versatility, these systems have been implemented for an ever-increasing number of industrial applications that may involve one or more systems characterized by complex structuring and possibly ill-defined behavior. The thrust has been here to enrich and complement the existing conventional techniques by adding to them a new generation of versatile techniques for identification and control based on connectionist modeling and learning. In fact, recent years have seen a surge in using artificial neural networks for nonlinear function approximation, for dynamic model representation, and for control. The reader may wish to consult [11] for an overview of the research work carried out in this regard. In what follows, an overview is made over conventional techniques for identification and adaptive control. Their features and limitations are highlighted. This is followed by presenting recent techniques based on connectionist modeling which have been implemented with success for systems identification and control of a wide range of industrial applications. The remaining part of the chapter deals with techniques recently developed to deal with chaotic systems and their time series prediction.

### 6.5.2 Conventional approaches for identification and control

A dynamic system is generally defined as a set of interconnected objects, in which a variety of time-varying variables interact to produce observable

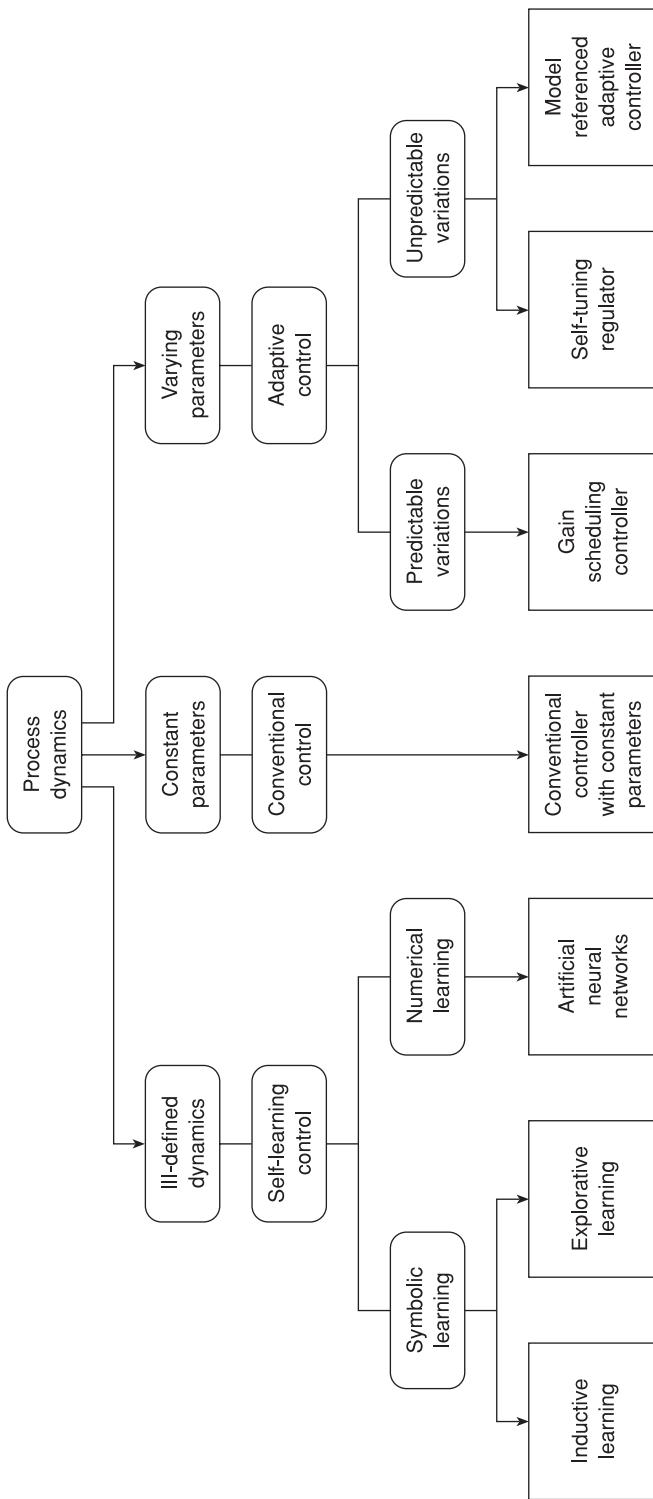
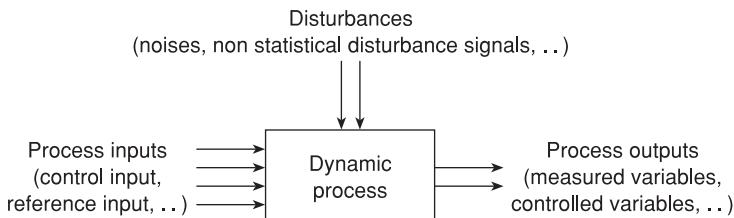


Figure 6.8: Classes of dynamic systems and their appropriate controllers



**Figure 6.9:** Block diagram of a dynamic system

outputs (Figure 6.9). A system may be subjected to external signals, which may be designer or environment-induced inputs.

To deal with the control aspect of a given system, a dynamic model is required. System models can be either analytical or experimental. The analytical models are based on the application of well-known basic physical laws [12], while experimental models are derived from the application of system identification tools [13, 14]. In some cases, particularly when the system structure is so complex that deriving an analytical model is difficult, experimental modeling techniques have been shown to be quite helpful in identifying the dynamic behavior of the system. This can also serve as a platform for designing adaptive controllers that are capable of providing the desired behavior, even under structural parametric variations of the system [14]. Next, a brief overview of the most frequently used experimental modeling techniques (extracted from experimental measurements) and adaptive control procedures is provided.

#### 6.5.2.1 Systems identification

The main goal in system identification is to establish an adequate dynamic model of the system such that the parameters, which provide the best fit between the measured input-output data and the model-generated values, are estimated. Model structure as well as the model parameters may be identified. The two well-known techniques for system identification are the nonparametric and the parametric estimation methods.

In nonparametric system identification, the goal is to obtain an input-output representation for which the dynamics are *a priori* known to be linear and time invariant. The method is known as nonparametric since it does not involve a parameter vector for use in the search for the best dynamical model. The nonparametric, time-domain method is useful when the designer has access to the step or impulse response of the system. The nonparametric frequency-domain-based method is essentially based on the collection of frequency response data when the system is excited by input signals of different frequencies. The method has the advantage of being simple in terms of data collection; it is nevertheless more time consuming to implement than its time-domain counterpart.

Another approach for system identification is the one based on the parametric estimation procedure. This method is particularly effective in providing

on line information on given process parameters and permits the designing of controllers that can adapt to process-dynamic changes while tracking a desired response. The well-known least-squares parameter estimation technique is among the most used approaches. It is based on the least squares principle, which states that the unknown parameters of a mathematical model are chosen to minimize the sum of the squares of the differences between the actually observed values and the computed ones. For more detailed analysis on this and other identification techniques, the reader may wish to consult [13], in which the theory is extensively covered, and [14–16], where a number of applications are provided.

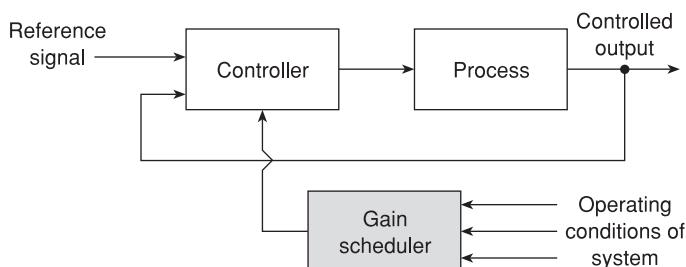
#### 6.5.2.2 Adaptive control

An adaptive control system uses a control scheme that is capable of modifying its behavior in response to changes in process dynamics. Adaptive controllers have been extensively used in several industries including chemical, aerospace, automotive, and pulp and paper, to name a few.

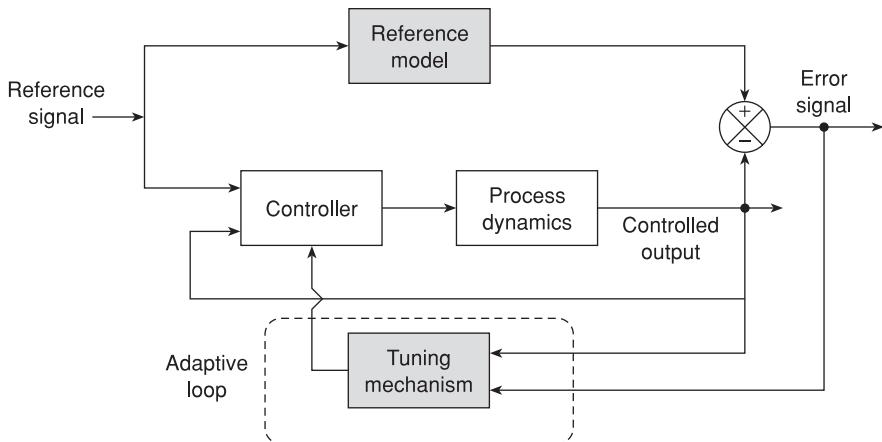
The rapid growth in the design of integrated and powerful information processors has made the use of adaptive controllers even more versatile. There are three well-known adaptive control schemes: gain scheduling, model-referenced adaptive control, and self-tuning regulators.

The gain scheduling technique is based on the adjustment of controller parameters in response to the operating conditions of a process [14]. This control scheme is particularly useful when the variations of the process dynamics are predictable. In fact, for a class of dynamic systems, it is possible that an explicit model of the system can be accurately described every time the operating conditions of the system take new values. Gain scheduling can be regarded as a mapping from the process parameters to the controller parameters. In practice, a gain scheduler can be implemented as a look-up table.

A block diagram of this adaptive scheme is shown in Figure 6.10. The two main drawbacks of this method are related to its *open-loop* behavior and to the discrete nature of the controller gains according to the actual operating conditions. Indeed, for intermediate operating conditions, no explicit control gains are assigned to the system, and the designer must apply interpolation techniques to avoid possible instabilities.



**Figure 6.10:** Block diagram of the gain scheduling controller



**Figure 6.11:** Block diagram of a model-referenced adaptive controller

The model-reference adaptive control (MRAC) is an adaptive control scheme particularly capable of handling processes with unpredictable changes [13]. This control procedure is based on the design of an adaptive scheme, whose objective is to drive the error signal between the response of the process and that of a given reference model to zero. The overall block diagram (Figure 6.11) consists of two main loops with different time constants. The inner loop, which is the faster one, is used for the regulation of the process, while the outer loop is designed for adjustment of the parameters of the inner loop regulator in order to drive the error signal to zero. Algorithms for designing the adaptation scheme for the adjustment-mechanism of the outer loop have been discussed in several textbooks and monographs and the reader may wish to consult references [14] and [16] for further details. Some instability problems for applying the MRAC procedure have been witnessed in some instances and remedies have been proposed to deal with them [16].

Self-tuning regulation is another adaptive control scheme characterized by its ability to handle dynamic processes that may be subjected to unpredictable changes in the system parameters [14]. A self-tuning regulator (STR) uses the outputs of a recursive identification scheme for plant parameters (outer loop) to adjust, through a suitable adaptation algorithm, the parameters of a controller located in the regulation loop of the system (inner loop), as shown in Figure 6.12. One can easily notice some similarities between STR and MRAC in terms of the inner and outer loop structuring. The main difference between the two schemes, however, is that while the STR design is based on an explicit separation between identification and control, the MRAC design uses a direct update of controller parameters to achieve asymptotic decay of the error signal to zero. In view of this fundamental difference in design, MRAC is referred to as a *direct adaptive control scheme*, while STR is known as an *indirect adaptive control scheme*.

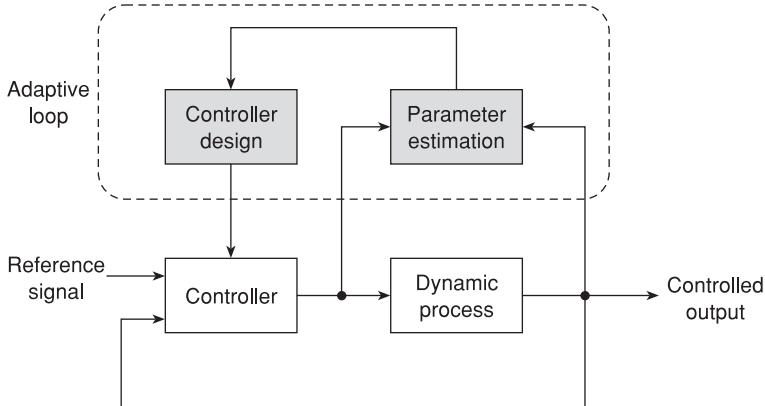


Figure 6.12: Block diagram of the self-tuning regulator

## 6.6 Neural network-based control approaches

The adaptive control techniques described above assume the availability of an explicit model for the system dynamics (as in the gain scheduling technique) or at least a dynamic structure based on a linear experimental model determined through identification (as in STR and MRAC). This may not be the case for a large class of complex nonlinear systems characterized by poorly known dynamics and time-varying parameters that may operate in ill-defined environments. Besides, conventional adaptive control techniques lack the important feature of learning. This implies that an adaptive control scheme cannot use the knowledge it has acquired in the past to tackle similar situations in the present or in the future. In other words, while adaptive control techniques have been used effectively for controlling a large class of systems with predefined structure and slowly time-varying parameters, they nevertheless lack the ability of learning and lack the ability for tackling the global control issues of nonlinear systems. Making assumption of linear structures of processes is not always possible and designers have to deal with the inherent nonlinear aspect of the systems dynamics.

As a potential remedy to some of these issues, designers have devised new control approaches that avoid the explicit mathematical modeling of dynamic processes. Some researchers have termed them as *expert intelligent control approaches*. Some of these approaches, such as those based on fuzzy logic theory, permit for an explicit modeling of the systems. Other approaches use well-defined learning algorithms through which the system is implicitly modeled and adapts autonomously its controller parameters so as to accommodate unpredictable changes in the system's dynamics. Here, the dynamical structure is not constrained to be linear, as is the case for most conventional adaptive control techniques. The study of these approaches has constituted a resurgent area of research in the last several years and several important results were obtained.

The family of controllers using fuzzy logic theory has been outlined in an earlier chapter. They are basically designed on the premise that they take full advantage of the (linguistic) knowledge gained by designers from past experiences. We tackle here another class of *intelligent controllers*, which are based on neural modeling and learning. They are built using algorithms that allow for the system to learn for itself from a set of collected training patterns. They have the distinctive feature of learning and adjusting their parameters in response to unpredictable changes in the dynamics or operating environment of the system. Their capability for dealing with nonlinearities, for executing parallel computing tasks and for tolerating a relatively large class of noises make them powerful tools for tackling the identification and control aspect of systems characterized by highly nonlinear behavior, time-varying parameters and possible operation within an unpredictable environment.

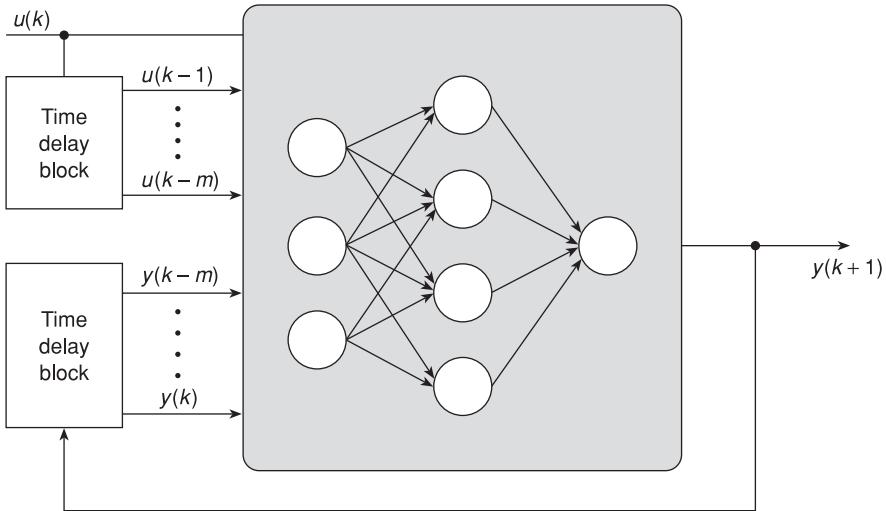
Given the fact that we deal here with dynamical models involving the states of the model at different time steps, it is only natural to design a specialized structure of neural networks that have the capability for “memorizing” earlier states of the system and for accommodating feedback signals. This is in contrast with the conventional neural networks (based on BPL) used mostly for system classification or function approximation. In fact, despite their proven capabilities as universal approximators, a limitation of standard feedforward neural networks, using backpropagation as the learning mechanism, is their limitations for exclusively learning static input-output mappings. While adept at generalizing pattern-matching problems in which the time dimension is not significant, systems with a non-stationary dynamics cannot be strictly modeled. This is the case of dynamic systems for which the representation is made through time-dependent states.

One way of addressing this problem is to make the network dynamic, that is, by providing it with a memory and feedbacks. A particular class of the recurrent structure is the so-called recurrent time-delay neural networks. One way for accomplishing this is to incorporate feedback connections from the output of the network to its input layer and include time delays into the NN structure through their connections. As there are propagation delays in natural neurobiological systems, the addition of these time delays stems from theoretical, as well as practical, motivations. Time-Delay Neural Networks (TDNNs) accomplish this by replicating the network across time. One can envision this structure as a series of copies of the same network, each temporally displaced from the previous one by one discrete time unit similar to the BPTT described earlier. The resultant structures can be quite large, incorporating many connections, and novel learning algorithms are employed to ensure rapid training of the network.

To illustrate the idea let us presume that the dynamic system input–output behavior of a nonlinear dynamic system is represented by the following equation:

$$y(k+1) = f[y(k), \dots, y(k-n); u(k), \dots, u(k-n)] \quad (6.9)$$

where  $y(k+1)$  represents the output of network at time  $(k+1)$ , and  $y(k), y(k-1), \dots, y(k-n)$  are the delayed output states serving as part of the



**Figure 6.13:** Time delayed recurrent neural network

network input along with the input signal  $u(k)$  and its delayed components:  $u(k - 1), \dots, u(k - n)$ . The network representation of such systems is depicted in Figure 6.13. Notice that this network is recurrent (feedback signals) with time-delayed inputs. While several algorithms have been proposed in the literature to deal with the training of time-delayed recurrent networks [17], the dynamic backpropagation algorithm proposed in [18] has been among the standard techniques used for this purpose. For the detailed derivation of this algorithm, one may wish to consult [11,17,18].

### 6.6.1 Neural networks for identification

As universal approximators, neural networks have been used in recent years as identifiers for a relatively wide range of complex dynamic systems (linear and nonlinear). Their capability for processing tasks in parallel and for tolerating noisy versions of the input signals make them excellent candidates for system identification. The process of systems identification, as mentioned previously, aims to find a dynamical model that approximates within a predefined degree of accuracy the actual plant dynamics when both systems are excited with the same signals. This means that we require a minimization of the error  $e(k + 1)$  between the predicted output  $\hat{y}_p(k + 1)$  and the actual output  $y_p(k + 1)$  of the system:

$$e(k + 1) = |\hat{y}_p(k + 1) - y_p(k + 1)|. \quad (6.10)$$

Since this error depends on the parameters of the network, the solution should provide the set of weights that would minimize it. Four major classes of models were proposed in the work of [18] and encompass a wide range of nonlinear input–output model representations:

- **Model 1:**  $y_p(k+1) = \sum_{i=0}^{n-1} a_i y_p(k-i) + g[u(k), \dots, u(k-m+1)]$
- **Model 2:**  $y_p(k+1) = f[y_p(k), \dots, y_p(k-n+1)] + \sum_{i=0}^{m-1} b_i u(k-i)$
- **Model 3:**  $y_p(k+1) = f[y_p(k), \dots, y_p(k-n+1)] + g[u(k), \dots, u(k-m+1)]$
- **Model 4:**  $y_p(k+1) = f[y_p(k), \dots, y_p(k-n+1); u(k), \dots, u(k-m+1)]$

In each of the models, the pair  $(u(k), y_p(k))$  represents the input-output pair of the identified plant at sample  $k$  and  $m \leq n$ .  $f$  and  $g$  are two smooth functions in their arguments. The characteristics of each one of the models are described in detail in the work of [18]. However, among the four models, the fourth one has been used most often given its sharing relevance to a wide majority of nonlinear dynamic systems. The choice for the appropriate learning algorithm for the neural identifiers (dynamic or static) depends mostly on whether the network takes as its inputs a delayed version of its output, or directly uses sample outputs from the plant itself. The two schemes that have been used most frequently are the *series-parallel* scheme and the *parallel* scheme. To illustrate the ideas, and if model four is taken as the system model of choice, the two corresponding schemes are then given as:

- **Series-parallel model**

The future value of the output  $\hat{y}_p(k+1)$  in this model is expressed as:

$$\hat{y}_p(k+1) = NNI[y_p(k), \dots, y_p(k-n+1); u(k), \dots, u(k-m+1)] \quad (6.11)$$

where  $NNI$  stands for mapping provided by the neural network identifier. This model is represented in Figure 6.14, and a careful inspection of its structure shows that while it still requires a set of delayed signals for its inputs, it does not involve feedback from the network output, which is the case for the second model known as parallel.

- **Parallel model**

The estimated future value of the output  $\hat{y}_p(k+1)$  in the parallel model is expressed as:

$$\hat{y}_p(k+1) = NNI[\hat{y}_p(k), \dots, \hat{y}_p(k-n+1); u(k), \dots, u(k-m+1)] \quad (6.12)$$

This model is illustrated in Figure 6.15 and uses the delayed recursions of the established output as some of its input.

One may notice here that given the particular structure of the series-parallel model, which doesn't involve recurrent states of the network output, the standard BPL could be used for extracting the parameters of the network. This is, however, not the case for the parallel structure given that the model includes a feedback loop including nonlinear elements. As such BPL cannot be applied in this case, and dynamic backpropagation would be the learning algorithm of choice.

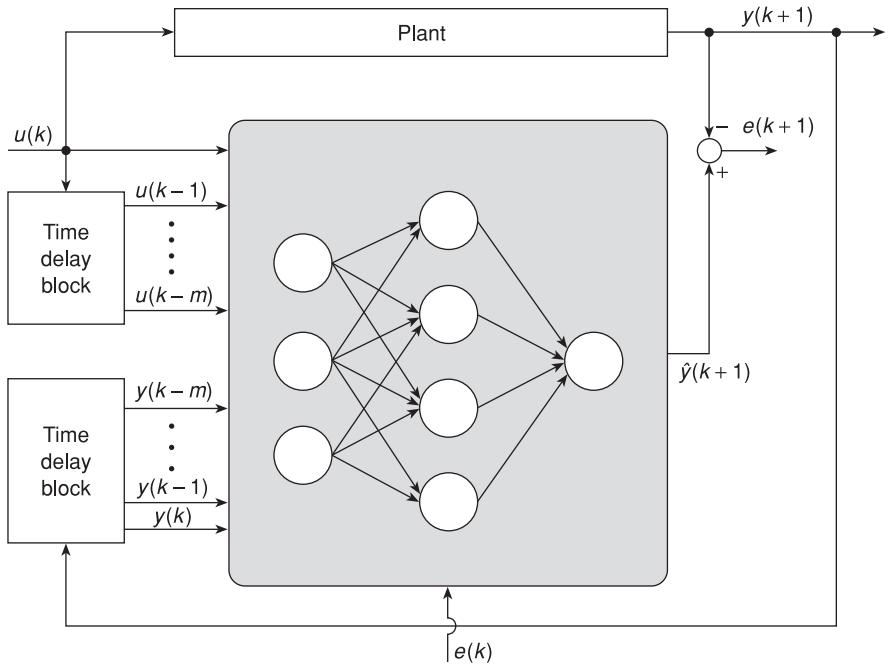


Figure 6.14: Series-parallel scheme for neural identification

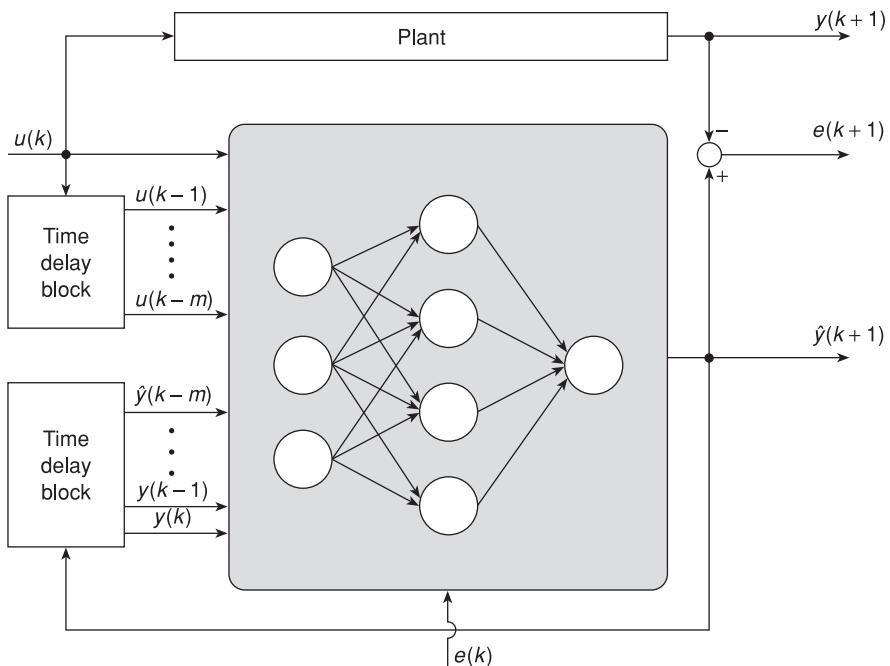


Figure 6.15: Parallel scheme for neural identification

### Example 6.1

The first step in controlling a process or system is to obtain knowledge about its dynamics. Using a recurrent network, a non-parametric model can be estimated using the system input–output data. Parametric identification can also be performed but the black-box (i.e., non-parametric) approach is used here as in [19]. In system identification, the physical process and the neural network are connected in parallel. The input and output data are collected, and then used to train the network until the error between the system output and network output reaches a specified level. Figure 6.16 illustrates the methodology.

In this example, a model of a simple pendulum system is identified using a recurrent network with the Elman architecture. Because of random initial conditions and the statistical nature of the training algorithms, solutions from one simulation to another can vary widely.

The system is represented in state space representation as:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -9.8 \sin x_1 - 0.5x_2 + 0.5u \end{cases} \quad (6.13)$$

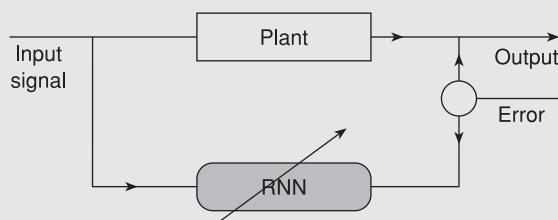
$$y = x_1 \quad (6.14)$$

where  $u$  is the input signal to the plant,  $x_1$  and  $x_2$  are the angle and the angular speed states respectively.  $y$  is the output variable. Note the time dependence is implicit for  $x_1$ ,  $x_2$  and  $u$ .

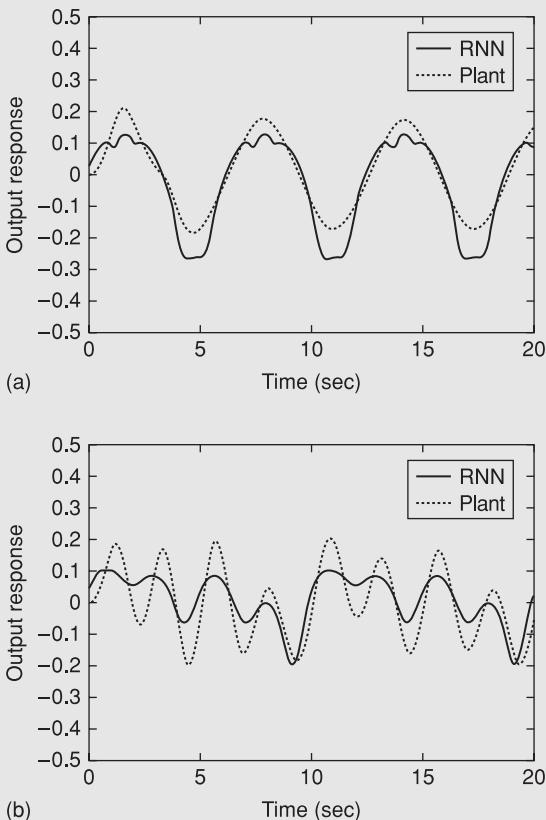
The training output is generated using the following input signal:

$$u(t) = \text{chirp}(0.1 \leq f \leq 1\text{Hz}) + \sin(t) + \sin(2t); \quad (6.15)$$

This particular input signal is chosen to allow the network to learn to generalize signals of varying amplitude and frequencies. The network with seven hidden neurons is trained for 400 epochs using as the performance index the sum of the squared errors (SSE). A performance index of 14.7 is achieved. Two validating sets of input signals are used to test how well the RNN has been trained to generalize the system and are given below:



**Figure 6.16:** System identification using RNN



**Figure 6.17:** (a) Identification results with first set of input. (b) Identification results with second set of input

$$u(t) = \sin\left(\frac{2\pi t}{2.5}\right) + \sin\left(\frac{2\pi t}{5}\right) + \sin\left(\frac{2\pi t}{10}\right) \quad (6.16)$$

$$u(t) = 3\sin(t) \quad (6.17)$$

Figure 6.17 shows the results of the trained RNN approximating the system response given the two test input signals. It is reasonable to assume that with a more optimal number of hidden nodes and a “richer” training data set, the performance can be improved. More experimentation with these parameters is necessary. Once the system has been approximately modeled, the control problem is later addressed.

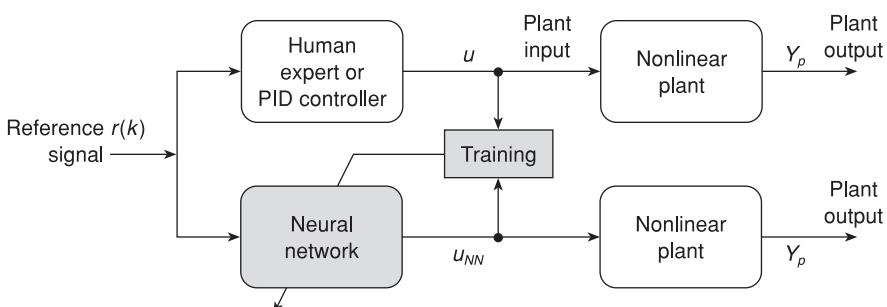
As a rule of thumb, to properly generalize the system, white noise should be used for training. The number of nodes in the hidden layer and the training algorithm chosen have an impact on the convergence performance. With more knowledge acquired in working with the methodology, one can expect to make better judgment in choosing these parameters.

## 6.6.2 Neural networks for control

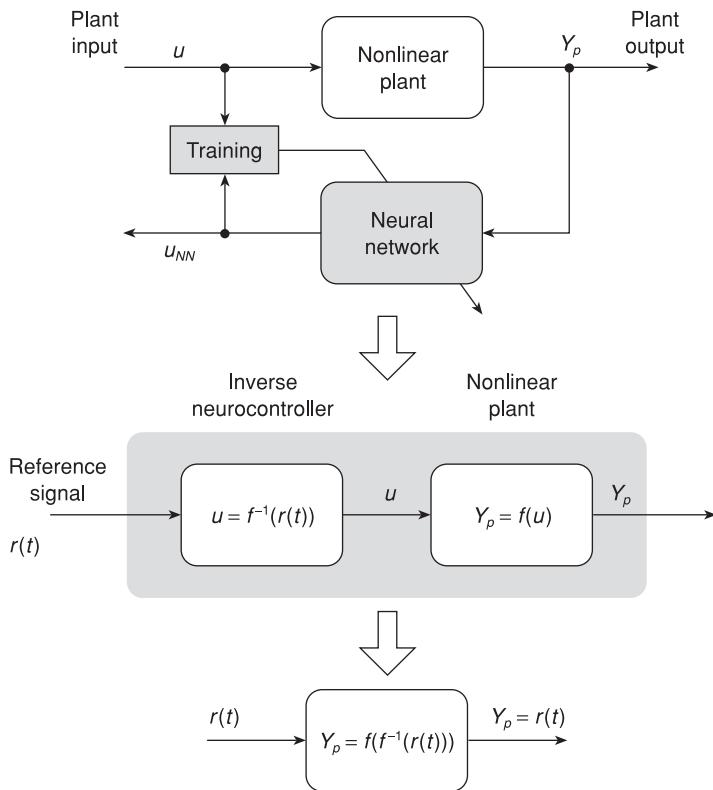
There have been a number of control schemes proposed in recent years involving neural networks for identification and control. We adopt here the classification provided in the work of Werbos [20], which suggests the following categories: *supervised* (“teacher” or model-based) *control*, *inverse model-based control*, and *neuro-adaptive control*. These categories are outlined next in more detail.

### 6.6.2.1 Supervised control

The first category assumes that a controller is synthesized on the basis of the knowledge acquired from the operation of the plant by a skilled operator (as shown in Figure 6.18) or through well-tuned PID controllers. During the nominal operation of the plant (at a particular operating condition) experimental data are collected from sensor devices, and are later used as a training set for a neural network. Once the training is carried out adequately, the process operator then becomes an upper-level supervisor without the need for being part of the control loop of the process. This is particularly useful in the case of hazardous environments or for improving the automation level of the plant. The neural network could also play the role of a gain interpolator of a set of PID controllers placed within the control loop of the plant. Once trained, the neural network will act here as a gain feeder for the PID controller, providing it with appropriate gains every time the operating conditions of the system change. This is a good alternative for interpolating between the gains instead of delivering them in a discrete way, as is the case of conventional gain scheduling. One of the main issues, however, is that the training data collected are usually corrupted with a large amount of noise and as such have to be filtered before becoming useful as valid training data. This has the potential of making of the neural network an expensive and possibly a time-consuming process. Another issue pertains to difficulties in extracting the knowledge acquired by the experts into a set of patterns that could be used for the network’s training.



**Figure 6.18:** Neural network acting as a supervisory controller



**Figure 6.19:** Neural network as inverse model-based controller

### 6.6.2.2 Inverse control

The second category where neural networks could be implemented within the control loop is known as *inverse control* or *inverse plant modeling*. Here, the designer seeks to build a neural network structure capable of mapping the input-output behavior of an inverse controller. The inverse controller is by definition a controller which, when applied to the plant, leads ideally to an overall transfer function of unity. A schematic representation of this control scheme is shown in Figure 6.19. Designing an *inverse neural controller* should be always done under the assumption that the process is minimum phase and causal. The main advantage of implementing an inverse controller into a neural structure is that it would allow for a faster execution and tolerance to a range of noises. The implementation of a neural network as an inverse controller, however, has been hindered by several difficulties, including the induction of unwanted time delays leading to possible discretization of processes that are originally continuous. This is mostly due to the effect of noncausality of the inverse controller as suggested in [21]. Moreover, the inverse controller cannot realistically match exactly the real plant, a fact that leads to unpredictable errors in the controller design.

### 6.6.2.3 Neuro-adaptive control

The third category of neural controllers pertains to implementing two dynamical neural networks within an adaptive control architecture, similar to the MRAC one. One of the networks serves here as an identifier, while the second serves as a controller. A large amount of research work has been dedicated to controllers belonging to this category [18]. Given the dynamic nature of the system being identified and controlled, recurrent time-delayed neural networks have been the tools of choice for this case. The main advantages of these neuro-adaptive structures pertain to their capability in effectively tackling the nonlinear behavior of the systems without compromising on their representation using linear approximations (such as the ARMA model auto-regressive moving average). This has not always been possible with the conventional schemes of adaptive control systems as described in earlier sections. As in the identification section, the controller here is handled using another structure of a time-delay neural network for which the output serves as input to the plant. The same output is delayed in time and fed back to the network to serve as part of its input signals. Now combining the two aspects (identification and control) within a well-defined adaptive structure such as the one described in a previous section leads to the representation of Figure 6.20. In this structure, which is very similar to the MRAC, identification is carried out first, and the identified model is then compared with the output of a reference model. The recorded error is then used to update the control law through modifications of the neural controller weight. This process is described in ample detail in [18]. Despite the fact that the structure is similar to the MRAC, the current scheme developed here is known as an *inverse neuro-adaptive control* scheme. This is mainly due to the fact that the error provided to the neural controller is not computed directly as the difference between the plant output and the reference, but rather between the model identified and the reference.

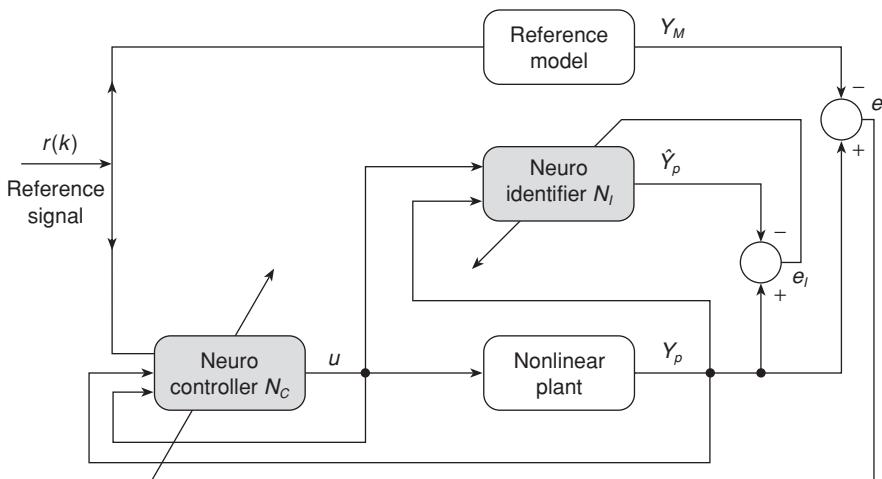


Figure 6.20: Neural network as neuro-adaptive controller

There exists in the literature a wealth of materials pertaining to neuro-control. In particular, some interesting problems tackled by Puskorius and Feldkamp [15] include the classical cart-pole problem, the bioreactor benchmark problem, and the engine idle speed control problem. RNNs, like their feedforward counterparts, can be challenging to train. Common pitfalls are poor local minima and the recency effect where there is a tendency for recent weight updates to cause the network to forget what it has learned (stability-plasticity dilemma) [18].

An excellent case study involving the stabilization of a high order power system is provided in the last chapter of the textbook and should serve the reader well in tackling complex aspects of identification and control of a nonlinear system using recurrent neural networks.

## 6.7 Dynamic neural networks for chaos time series prediction

Chaos is a relatively new branch of mathematics that seeks to explain the domain of highly nonlinear phenomena. The field originated from Lorenz' seminal paper on convective flows in a simple fluid model [22]. The field has resurfaced in recent years as a fundamental branch of mathematics, known as the mathematics of chaos, and for over 20 years the field has witnessed major interest from scientists and engineers. The mathematics of chaos has been applied in describing a wide variety of nonlinear systems, such as stock market price time series, dynamics of galaxy formation, weather forecasting, and optical characteristics in refracting laser light, just to name a few. As the understanding of chaotic systems grows, no doubt more natural and artificial systems will be identified as exhibiting chaotic behavior.

Unfortunately, due to their nonlinearities, the behavior of chaotic systems is notoriously difficult to predict and control. Recently, there has been intense research work on the problem of chaotic system control and prediction, with mixed results. Traditional techniques of system prediction and established methods of control have been shown to be lacking, and a wealth of new techniques have been created to deal with the inadequacies. These have ranged from extensions of traditional methods, to entirely new schemes employing the chaos inherent in the system itself, to aid in its control. Some of these approaches use the tools of computational intelligence for chaotic system prediction and control.

Computational intelligence seeks to apply methods that differ from traditional artificial intelligence methods, and includes such techniques as artificial neural networks (ANNs), fuzzy logic systems, genetic algorithms (GAs), and others. These have proven to be generally more flexible than other methods, while offering comparable or superior performance. We restrict our preview here of the application of neural networks to the control and prediction of chaotic systems. Prior to that, we briefly outline the historic developments behind chaos mathematics, and provide an overview of conventional approaches used to solve the problem.

### 6.7.1 Background

The theory behind the branch of mathematics known as chaos theory is barely five decades old and owes its inception to Lorenz, who first published his findings in a seminal paper describing a simple model of convective fluid flow [23]. At first, research in the field was slow to get off the ground but has seen rapid advances in recent years as chaotic systems are rapidly coming to the forefront of scientific research. This was associated with the huge advances made in simulating chaotic systems on powerful computers. Chaotic systems have interesting properties, such as the ability to alter their output drastically with small changes in input. Hence, concurrent with the growth of research into chaotic systems has been a growth in the desire to predict and control them.

All chaotic systems are nonlinear and deterministic. The implications of this are many, but, first and foremost, random effects or noise do not cause chaos. Rather, extreme nonlinearities in the system cause it to appear random or noisy since even small changes in the input have large effects on the output. However, chaotic systems are deterministic, such that perfect knowledge of the input will still yield a perfectly predictable output. Unfortunately, in real-world applications such perfect knowledge is rarely, if ever, attainable, so prediction or control of chaotic systems remains an important problem despite the deterministic nature of the domain.

While chaotic systems abound in nature and are evident in astrophysics, economics, ecology, and meteorology, to name a few, two time series with chaotic behaviour rate particular mention since they are quite prevalent in control and prediction literature. The first is the *logistic map* [24], famous not only for its surprisingly complex behavior, but also as a historically important equation in the development of chaos theory. It is given by a deceptively simple difference equation

$$x_n = rx_{n-1}(1 - x_{n-1}) = f_r(x_n) \quad (6.18)$$

where  $r$  is a numerical parameter, with  $x$  being between 0 and 1. The logistic map was historically used as a gross simplification of biological population growth and decay.

The second time series is the delay-differential equation, known as the *Mackey–Glass differential equation*, introduced in [25], and is dynamically represented as:

$$\frac{dx}{dt} = \frac{ax(t - \tau)}{1 + x^c(t - \tau)} - bx \quad (6.19)$$

where  $a$ ,  $b$ ,  $c$ , and  $\tau$  are parameters of the series. This model has also been used for biological system modeling involving time delays.

### 6.7.2 Conventional techniques for chaos system prediction and control

Although unbeknownst to him at the time, Yule [26] performed perhaps the earliest attempt at chaotic system prediction by attempting to predict sun

spot cycles. He used the then novel approach of using previous data in the time series to predict subsequent values, rather than attempting to linearize the system. Chaotic system prediction has come a long way since then, albeit mostly in the last 15 years, and various techniques have been implemented with varying degrees of success [27]. The technique of Farmer and Sidorowich [27] attempts to embed a time series in a state space, which then allows them to make short-term predictions. In [28], Pyragas, citing limitations to the subsequently described OGY method [29], devised two methods to stabilize unstable periodic orbits. The first uses an external oscillator combined with system feedback to provide a controlling input, while the second has only a delayed self-controlling feedback. Additionally, Fowler compared the behavior of chaotic systems with that of noise-corrupted ones, and proposed the use of stochastic control techniques [30]. His technique uses a static estimator similar to a Kalman filter.

In 1990, Ott, Grebogi and Yorke [31] developed the OGY method of chaotic system control, which applies a series of small perturbations to push chaotic systems towards stable periodicity. This method distinguished itself from others at the time because *a priori* knowledge of the system was not required. It is perhaps the most prevalent conventional technique for chaotic system prediction and control. For a collection of papers describing recent applications of the OGY technique to chaotic system control and prediction, please refer to the excellent survey by Shinbrot, Grebogi, Ott and Yorke [32].

As previously noted, the original technique suffered from some limitations. It was limited to one- or two-dimensional maps, or to a limited number of control parameters. These limitations have been addressed in subsequent work [33], although the technique is still only applicable when the system is close to the fixed point of interest. This limitation has led others to pursue alternate avenues of control and prediction.

### 6.7.3 Artificial neural networks for chaos prediction

As mentioned previously, one of the first approaches to controlling nonlinear dynamic systems using neural networks was performed in 1990 by Narendra and Parthasarathy [34]. Their approach used both feedforward structures and recurrent structures, two major architectures discussed previously. Following their lead, other researchers have devised other approaches. While it is beyond the scope of this short overview to detail all the approaches that used feedforward networks for chaos prediction, due to their sheer number and diversity, it is possible to group them loosely by type and functionality.

#### 6.7.3.1 Conventional feedforward networks

Approaches using standard (static) feedforward networks for the control and prediction of chaotic systems are described in ample detail in [35] in a plethora of methods by which the task was accomplished. Most of the methods use novel network topologies, improved or different learning rules,

or newly developed transfer functions. Examples of the latter alterations are the wavelet networks [36], which use wavelet functions for the transfer functions. Platt created a resource-allocating network that added nodes whenever new patterns were presented to it [37], thus creating a self-growing one. In [38], a network trained with supervised learning used a conventional controller as its teacher; this suggests that this particular network cannot outperform the conventional controller. In [39], Choi and Choi introduced a piecewise linear method that was then implemented for parallel computation in a neural network, applied to the prediction of  $\text{NH}_3$  laser intensities. In [40], a comparison is made between feedforward networks trained with backpropagation learning and auto-regressive moving average (ARMA) numerical models, concluding that chaotic series with large positive Lyapunov exponents can only be predicted by the connectionist approach. King and Nyman [41] used a neural network to model an electric arc furnace and compare the performance of the system against that of human heuristic control. It was noted in [42] that outliers, outputs with large errors from the desired signals, detract from the robustness of network predictors. To deal with this problem, a new influence function, the mean log squared error (MLSE), was used during backpropagation learning. Flake [43] showed that it was possible to control a chaotic system with a simple linear perceptron.

As mentioned previously, radial-basis function (RBF) networks arise from considering the problem of neural network design as an approximation problem in a high-dimensional space. Learning then becomes the problem of minimizing some error function over a multidimensional surface that approximates the training data, and generalization becomes interpolation of the test data. In this context, the hidden kernel functions provide a basis when inputs are expanded into a functional space.

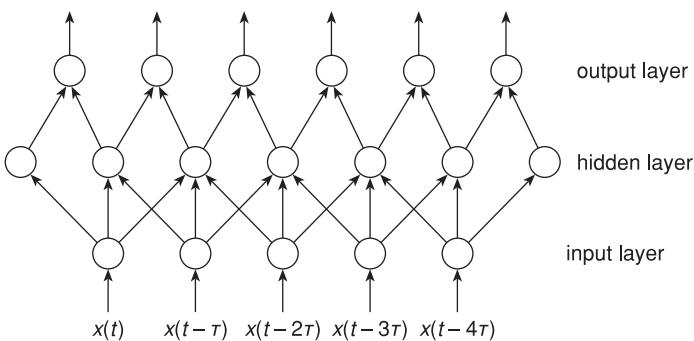
Radial basis function techniques for the control and prediction of chaotic systems [44] are somewhat less varied in their methods than the other neural network approaches. Nonetheless, there are numerous applications of the technique that have shown considerable promise. Lapedes and He created a successive approximation RBF network that started with a coarse approximation and then combined them to achieve a more accurate prediction [45]. In [46], a contrasting approach was used: RBF networks were created to be as simple as possible, with performances that are “close to” the minimum solution being good enough. An RBF network was applied to the logistic map prediction problem [47] and improved by adding a cost function that not only examined the prediction error at each sample, but also the smoothness of the function that was used. It was shown that the smoothed network had better performance than the ordinary RBF network. Dogaru *et al.* implemented in [48] a modified RBF network in VLSI, for the prediction of chaotic time series. A different approach using a recursive application of the Gram–Schmidt process [49] for the training of an RBF network was compared with He and Lapedes’ model, as well as against a nonrecursive training model. In the same vein, Blanzieri and Katenkamp [50] compared two well-known learning methods with the goal of on line learning.

### 6.7.3.2 Recurrent neural networks (RNNs)-based predictors

Recurrent neural networks (RNNs), or chaotic neural networks as they are sometimes referred to, represent a fundamental change from the philosophy of feedforward ANNs. RNNs as mentioned earlier have recurrent connections between nodes from one layer to those in previous ones. Additionally, self-recurrent connections are allowed to exist, in which output from a node is passed back as input to the node itself. These recurrent connections endow RNNs with two specific properties: first, it means that RNNs possess a rudimentary memory, since activation is stored in the recurrent connections which then pass it back on the next calculation. Second, since network outputs depend in part on previous ones, RNNs are capable of exhibiting internal chaotic behavior, which is the source of their second name. It is unlikely that the chaotic behavior of the network itself helps it to predict and control chaotic systems. Nevertheless, RNNs have been applied successfully to the problem for some time, and are amongst the more commonly used approaches.

Approaches that used some form of recurrent structure are described in [51] and again show a variety of different methods. RNNs are used less often than feedforward structures, but are generally more recent and as such present a possible trend in the use of NNs for the control and prediction of chaotic systems. In [51], adaptive time constants were introduced to each of the nodes in the network, creating a dynamic recurrent structure. Rao and Kumthekar [52] used the novel wavelet transfer functions evidenced in some feedforward networks, and applied them to recurrent networks. Other researchers compared the performance of recurrent structures and feedforward structures, for the prediction of chaotic time series, as early as 1993 [53]. In [54], a novel training scheme was used where the series following the training value was sampled, rather than simply using the next value in the series, so that the network would be extracting the relative “periodicity” of the time series, rather than simply the next value. Hwang and Little [55] showed that a recurrent system that implicitly derived models of the statistical parameters of a time series outperformed an explicit method for the same task. An interesting approach in [56] used a recurrent network where each neuron used an ARMA process, which was successfully applied to the prediction of the Mackey–Glass time series. Recently, Teran *et al.* [57] compared a dynamically recurrent neural network with a static one for the prediction of chaotic time series, finding that dynamic behavior allowed the network to more accurately predict series. For more information on dynamic networks, which are often closely related to recurrent structures, one may refer to the earlier sections on dynamic neural networks.

Another class of recurrent neural networks using time delays has been proposed for prediction of chaotic time series. They are known as time-delay neural networks (TDNN). It appears that TDNNs are capable of developing shift-invariant internal representations of the input, which are then used for optimal generalization of the output. Naturally, this appears to be beneficial for predicting and controlling chaotic systems, whose time dependence is largely unknown. Figure 6.21 shows a simple time-delay neural network.



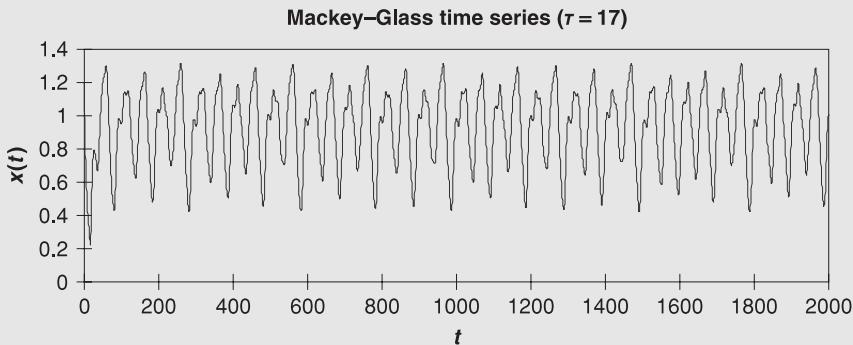
**Figure 6.21:** Model of single-input time-delay neural network. Inputs are successively delayed by  $\tau$  units, with hidden layer elements depending on current and two delayed inputs. Outputs depend on two delay units only

An embodiment of the TDNN is a multilayer perceptron in which each connection is represented by a finite-duration impulse response (FIR) filter. These networks, closely related to TDNNs, are called FIR multilayer perceptrons, or simply FIR networks. These have also been used for the task at hand. They are feedforward in structure, but achieve temporal properties due to the structure of their connections.

Dynamic networks have been applied to the problem of control and prediction in a few cases [58], although they do not show as much variety as with the other methods. In particular, widespread use of the FIR network does not appear to be the case. Lin, Dayhoff and Ligomenides [59] used an adaptive time-delay neural network (ATNN) to predict chaotic time series, and then compared it to other approaches using static time-delay neural networks and regular backpropagation networks. More research into the ATNN, by the same researchers, is summarized in [60]. Contrasting their work, Day and Davenport wrote an extension to backpropagation for continuous-time feed-forward networks [61]. In [62], Wan details the FIR network architecture, which was entered into a competition for the prediction of chaotic time series. Gilli took a different approach and created a dynamic network structure based on cellular networks [63].

### Example 6.2

To illustrate the ideas and to verify the effectiveness of each method outlined here and to provide some sort of comparison between the different techniques, the prediction of the Mackey–Glass chaotic time series was chosen as a benchmark problem. The Mackey–Glass delay differential equation was provided earlier. The Runge–Kutta ( $O^4$ ) technique was used to generate the input–output pairings, over 2000 time units, with a step size of 1. The numerical parameters of the equation were set to  $a = 0.2$ ,  $b = 0.1$ ,  $c = 10$ , and  $\tau = 17$ , as has been the



**Figure 6.22:** Mackey–Glass time series

case in many of the references. The initial conditions  $x(0) = 1.2$ , and  $x(t < 0) = 0$ , were used for the approximation as well, and the resulting time series is shown in Figure 6.22. The task was then to construct the different methods, for the prediction of the time series at  $x(t + 6)$ , in a way so as to create an objective comparison between them. Further, different performance measures had to be generated in order to compare the techniques, since a method that predicts with errors half as large as that of another method might not be better if it takes twice as long to implement or to train; the key is in the application that is required.

A simple connectionist model, in the form of a feedforward neural network using backpropagation learning, was implemented. This was a three-layer structure with four inputs, 24 hidden units and one output. Here, closely following other researchers, the four inputs represented the time series for  $x(t - 6c)$ , for  $c = \{0, 1, 2, 3\}$ . Training was performed over 20,000 epochs, with a learning rate parameter of 0.3, with the first 500 samples in the series (after discarding the initial 200 to account for transient effects) being used as the training set. The termination error tolerance was set to 0.001, but the network failed to converge to this value during the allotted training time. The final 1300 data points were used as the test set, and the network nonetheless managed an RMSE of 0.011, a much better performance than some conventional techniques. Application of the non-dimensional error index generated a value of 0.049.

Since radial-basis function neural networks are more suited for the task of function approximation than simple backpropagation learning ones, two radial basis function neural networks were constructed next. The first structure had four inputs, 24 hidden units, and one output. This was used mainly as a comparison of the prediction performance between the backpropagation learning network and a similar RBF network. Using the same training and test data, the first RBF network converged to an RSME of only 0.0083 (non-dimensional index 0.036). This shows that radial-basis function networks are more effective at the task than the more general backpropagation networks. Next, in an attempt to minimize the prediction errors with this technique, a network with 100 hidden units was constructed. This had little additional effect on the performance, but

did decrease the RSME to 0.0038, with a non-dimensional index of 0.016. The extra computation time to train this larger network was not particularly significant, yet the improvement in the results does not seem to merit the additional time (Table 6.1).

**Table 6.1: Results of applying different methods**

Method	Training cases	RSME	Non-dimensional error index
Auto regressive model	500 <sup>1</sup>	0.043 <sup>2</sup>	0.19 <sup>1</sup>
6th-order polynomial	500	0.0091	0.040 <sup>1</sup>
Cascaded-correlation NN	500	0.014	0.060 <sup>1</sup>
Feedforward NN (backpropagation)	477	0.011	0.049
RBF NN (24 hidden nodes)	477	0.0083	0.036

<sup>1</sup> Obtained from [53].

<sup>2</sup> Obtained by multiplying the non-dimensional error index from [53] by the Mackey–Glass standard deviation.

## 6.8 Summary

The research in the area of recurrent neural networks has been very active for the past 10 years. The discussion of the architectures, training techniques and applications gives an overview of the topic. As researchers continue to lay the theoretical foundation and develop efficient training algorithms for the next-generation applications, one is always reminded of the difficult past of neural network development. One should not exaggerate the potential of any methodology, which, in this case, is the RNNs, but to discern with a scientific eye the line between their capabilities and limitations. It is perhaps through the successful application of recurrent networks to real-world problems that researchers can justify time and effort spent working in this relatively new field.

## Problems

1. Represent in a diagram the Elman network (using the Elman architecture) of the dynamic system whose equation is given in example 6.1. Draw its feedforward equivalent using the same procedure as developed for the BPTT of section 6.3.1.
2. The dynamics of a time series is represented by the following nonlinear difference equation:

$$y(t) = (0.3 - 0.6 \exp(-y^2(t-1)))y(t-1) - (0.8 + 0.9 \exp(-y^2(t-1)))y(t-2) + 0.3 \sin(\pi y(t-1))$$

- (a) Considering  $y(t)$  as the output of the time series and  $y(t-1)$  and  $y(t-2)$  as its inputs with  $(y(t-1), y(t-2)) \in [-1.8, 1.8]^2$  display the surface output. Note the linear behavior of  $y(t)$  as a function of  $y(t-2)$  when  $y(t-1)$  is held at 0.

- (b) Design a feedforward neural network to estimate the output surface. You may wish to use 500 sample data for learning and testing purposes.
- (c) Compare the results obtained in (b) with those you obtain using an RBF network with 30, 60, 90 hidden nodes, respectively. Comparison is made in terms of RMSE.
3. We need to identify the nonlinear dynamics of a system represented by the following difference equation:

$$y_p(k+1) = \frac{y_p(k)}{1 + y_p(k)} + g(u(k))$$

This system belongs to the family of processes represented by model 3. We identify this system using a series-parallel identification scheme. Let us presume here that  $g(u(k)) = u^2(k)$ .

- (a) Represent a diagram similar to Figure 6.14 outlining the series-parallel nature of this system.
- (b) With the input  $u(k)$  belonging to  $[-1, 1]$ , find the interval of variation.
- (c) Design an RBF neural network to identify the input dynamics  $g(u(k))$  and the mapping  $\frac{y_p(k)}{1 + y_p(k)}$ .
- (d) Identify the two mappings of (c) using a BPL-based NN.
- (e) Run simulation results to compare the original plant with the identified one using an input  $u(k)$  given by:

$$u(k) = \frac{1}{2} \sin\left(\frac{2\pi k}{12.5}\right) + \frac{1}{2} \sin\left(\frac{2\pi k}{25}\right)$$

4. (a) Repeat problem 3 when  $g(u(k))$  is given by:

$$\begin{aligned} g(u(k)) &= u^3(k) \\ g(u(k)) &= u^2(k) + u^3(k) \end{aligned}$$

- (b) For  $g(u(k)) = u^2(k) + u^3(k)$ , can one use the results obtained in problem 3? What are the precautions that have to be taken there?
5. Classify the following nonlinear systems according to the models to which they belong:

- (a)  $y_p(k+1) = \frac{y_p(k)}{1 + y_p(k-1)} + u(k)$
- (b)  $y_p(k+1) = \frac{y_p(k)u(k)}{1 + y_p(k-1)}$
- (c)  $y_p(k+1) = 0.2y_p(k) + 0.7y_p(k-1) + y_p(k-1)u(k)$
- (d)  $y_p(k+1)u(k) = 0.7y_p(k) + 0.2y_p(k-1)$

6. We need to predict the Henon Series using a feedforward neural network-based structure. The series is represented in state space as:

$$\begin{cases} x_{n+1} = 1 - \alpha x_n^2 + y_n \\ y_{n+1} = \beta x_n \end{cases}$$

- (a) Display the iterated time series  $x_n$  (as function of time) when  $\alpha = 1.4$  and  $\beta = 1.3$  (for 200 iterations).

- (b) Display the phase plot ( $x$  vs.  $y$ ) and deduce that it represents the behavior of a “strange attractor.”
- (c) Design a feedforward neural network with one hidden layer to predict the behavior of the Hénon time series (in time domain). Show that the neural network-based prediction starts losing performance at around the tenth iteration and explain why.

## References

1. Kolen, J.F., and Kremer, S.C. (2001) *A Field Guide to Dynamical Recurrent Networks*, IEEE Press.
2. Orr, G., Schraudolph, N., and Cummins, F. (Fall 1999) *Lecture Notes for CS-449: Neural Networks*, <http://cognet.mit.edu/MITECS/Entry/jordan>, Willamette University.
3. Lin, C.T., and Lee, C.S. (1996) *Neural Fuzzy Systems*, Prentice Hall Publishing.
4. Dos Santos, E.P., and Von Zuben, F.J. (1999) “Improved second-order training algorithms for globally and partially recurrent neural networks,” *International Joint Conference on Neural Networks (IJCNN)*, vol. 3, pp. 1501–6.
5. De Jeses, O., and Hagan, M.T. (2001) “Backpropagation through time for a general class of recurrent network,” *Proceedings on International Joint Conference on Neural Networks (IJCNN)*, vol. 4, pp. 2638–43.
6. Atiya, A.F., and Parlols, A.G. (May 2000) “New results on recurrent network training: unifying the algorithms and accelerating convergence,” *IEEE Transactions on Neural Networks*, vol. 11, issue 3, pp. 697–709.
7. Wern-Jun Wang, Yuan-Fu Liao, and Sin-Horng Chen (2002) “RNN-based prosodic modeling for mandarin speech and its application to speech-to-text conversion,” *Speech Communication*, vol. 36, pp. 247–65.
8. Kermanshahi, B. (1998) “Recurrent neural network for forecasting next 10 years loads of nine Japanese utilities,” *Neurocomputing*, vol. 23, pp. 125–33.
9. Maciejowski, J.M. (1989) *Multivariable Feedback Design*, Reading, MA: Addison-Wesley, pp. 265–323.
10. Lewis, F. (1992) *Applied Optimal Control and Estimation*, Englewood Cliffs, NJ: Prentice-Hall, pp. 2526–72.
11. Gupta, M., and Rao, D. (1995) *Intelligent Control*, IEEE Press.
12. Shearer, J.L., Murphy, A.T., and Richardson, H.H. (1971) *Introduction to System Dynamics*, Reading, MA: Addison-Wesley.
13. Ljung, L. (1988) *System Identification-Theory for the User*, Englewood Cliffs, NJ: Prentice-Hall, pp. 20–45.
14. Astrom, K., and Wittenmark, B. (1995) *Adaptive Control*, 2nd ed., Reading, MA: Addison-Wesley, pp. 3–63.
15. Franklin, G., Powell, D., and Emami-Naeini, A. (1995) *Feedback Control of Dynamic Systems*, 3rd ed., Reading: MA, Addison-Wesley, pp. 144–50.
16. Sastry, S., and Bodson, M. (1989) *Adaptive Control: Stability Convergence and Robustness*, Englewood Cliffs, NJ: Prentice-Hall, pp. 10–45.
17. Haykin, S. (1994) *Neural Networks: A Comprehensive Foundation*, Englewood Cliffs, NJ: McMillan College Publishing Company.
18. Narendra, K., and Parthasarathy, K. (1990) “Identification and control of dynamical systems using neural networks,” *IEEE Transactions On Neural Networks*, vol. 1, no. 1, pp. 4–27.

19. Tsoukalas, L.H., and Uhrig, R.E. (1997) *Fuzzy and Neural Approaches in Engineering*, John Wiley & Sons, Inc.
20. Werbos, P. (1990) "Neuro control and related techniques," *Handbook of Neural Computing Applications*, New York, Academic Press, pp. 345–80.
21. Psichogios, D., and Ungar, L. (1991) "Direct and indirect model based control using artificial neural networks," *Industrial and Engineering Chemistry Research*, vol. 30, no. 12, pp. 2564–73.
22. Lorenz, E.N. (1963) "Deterministic nonperiodic flow," *Journal of the Atmospheric Sciences*, vol. 20, 131–41.
23. Hakim, N.Z., Kaufman, J.J., Cerf, G., and Meadows, H.E. (1990) "A discrete-time neural network model for systems identification," *International Joint Conference on Neural Networks*, vol. 3, 593–8.
24. Platt, J. (1991) "A resource-allocating network for function interpolation," *Neural Computation*, vol. 5, no. 2, 213–25.
25. Deppisch, J., Bauer, H.-U., and Geisel, T. (1991) "Hierarchical training of neural networks and prediction of chaotic time series," *Physics Letters A*, vol. 158, nos 1–2, 57–62.
26. Weigend, A.S., Huberman, B.A., and Rumelhart, D.E. (1990) "Predicting the future: a connectionist approach," *International Journal of Neural Systems*, vol. 1, no. 3, 193–209.
27. Rogers, S.K., and Kabrisky, M. (1989) "1988 AFIT neural network research," *Proceedings of the IEEE 1989 National Aerospace and Electronics Conference*, vol. 2, 688–94.
28. Matsuura, T., Hiei, T., Itoh, H., and Torikoshi, K. (1995) "Active noise control by using prediction of time series data with a neural network," *1995 IEEE International Conference on Systems, Man and Cybernetics*, vol. 3, 2070–5.
29. Majetic, D. (1995) "Dynamic neural network for prediction and identification of nonlinear dynamic systems," *Journal of Computing and Information Technology*, vol. 3, no. 2, 99–106.
30. Navone, H.D., and Ceccatto, H.A. (1995) "Learning chaotic dynamics by neural networks," *Chaos, Solitons and Fractals*, vol. 6, 383–7.
31. Diambra, L., and Plastino, A. (1995) "Maximum entropy, pseudoinverse techniques, and time series predictions with layered networks," *Physical Review E*, vol. 52, no. 4, 4557–60.
32. Mulpur, A., Mulpur, S., and Bongseog, J. (1995) "Control of chaotic oscillations using neural networks," *Proceedings of the 4th IEEE Conference on Control Applications*, 560–5.
33. Konishi, K., and Kokame, H. (1995) "Stabilizing and tracking unstable focus points in chaotic systems using a neural network," *Physics Letters A*, vol. 206, nos 3–4, 203–10.
34. Narendra, K.S., and Parthasarathy, K. (1990) "Identification and control of dynamical systems using neural networks," *IEEE Transactions on Neural Networks*, vol. 1, no. 1, 4–27.
35. Kohlmorgen, J., Muller, K.-R., and Pawelzik, K. (1994) "Competing predictors segment and identify switching dynamics," *Proceedings of the International Conference on Artificial Neural Networks*, vol. 2, 1045–8.
36. Der, R., and Herrmann, M. (1994) "Nonlinear chaos control by neural nets." *Proceedings of the International Conference on Artificial Neural Networks*, vol. 2, 1227–30.
37. Srivastava, A.N., and Weigend, A.S. (1994) "Conditional density estimation for physiological control systems," *Proceedings of the 16th Annual International*

- Conference of the IEEE Engineering in Medicine and Biology Society, vol. 2, 1073–4.
38. Choi, J.Y., and Choi, C.-H. (1995) “Continuous piecewise linear fitting approach for nonparametric regression,” *Proceedings of Neural, Parallel and Scientific Computations*, vol. 1, 102–5.
  39. Arena, P., Fortuna, L., and Xibilia, M.G. (1994) “Predicting complex chaotic time series via complex-valued MLPs,” *1994 IEEE International Symposium on Circuits and Systems*, vol. 6, 29–32.
  40. Otawara, K., and Fan, L.T. (1995) “Controlling chaos with an artificial neural network,” *Proceedings of the 1995 IEEE International Conference on Fuzzy Systems*, vol. 4, 1943–8.
  41. Welstead, S. (1994) “Predicting prediction: error bounds for feedforward neural networks,” *World Congress on Neural Networks*, vol. 2, 328–3.
  42. Kil, R.M., and Choi, J.Y. (1994) “Online learning of a network with Gaussian kernel functions,” *World Congress on Neural Networks*, vol. 2, 321–7.
  43. Lehtokangas, M., Saarinen, J., Huuhtanen, P., and Kaski, K. (1993) “Chaotic time series modeling with optimum neural network architecture,” *Proceedings of the 1993 International Joint Conference on Neural Networks*, vol. 3, 2276–9.
  44. Elsner, J.B. (1991) “Predicting time series using a neural network as a method of distinguishing chaos from noise,” *Proceedings of the 1991 International Conference on Neural Networks*, 145–50.
  45. Gardner, S. (1991) “Polynomial neural networks for signal processing in chaotic backgrounds,” *International Joint Conference on Neural Networks*, vol. 2, 890.
  46. Gardner, S. (1991) “Polynomial neural nets for signal and image processing in chaotic backgrounds,” *Proceedings of the SPIE – The International Society for Optical Engineering*, vol. 1567, 451–63.
  47. Chan, D.Y.C., and Prager, D. (1991) “Analysis of time series by neural networks,” *IEEE International Joint Conference on Neural Networks*, vol. 1, 355–60.
  48. Coughlin, J.P., and Baran, R. (1991) “Time series prediction with linear and nonlinear adaptive networks,” *IEEE International Joint Conference on Neural Networks*, vol. 1, 379–84.
  49. Ginzberg, I., and Horn, D. (1991) “Learnability of time series,” *IEEE International Joint Conference on Neural Networks*, vol. 3, 2653–7.
  50. Pethel, S.D., and Bowden, C.M. (1992) “Neural network applications to nonlinear time series analysis,” *Proceedings of the 1992 International Conference on Industrial Electronics, Control, Instrumentation, and Automation*, vol. 2, 1094–8.
  51. De Souza Jr, M.B., Pinto, J.C., and Lima, E.L. (1993) “Neural net-based model predictive control of a chaotic continuous solution polymerization reactor,” *Proceedings of the 1993 International Joint Conference on Neural Networks*, vol. 2, 1777–80.
  52. Szu, H., and Henderson, R. (1993) “Control of chaos using neural networks,” *Proceedings of the 1993 International Joint Conference on Neural Networks*, vol. 2, 1781–4.
  53. Shimodaira, H. (1995) “A method for selecting similar learning data based on correlation coefficients in the prediction of time series using neural network,” *Transactions of the Information Processing Society of Japan*, vol. 36, no. 2, 266–74.
  54. Narayan, S., and Page, E.W. (1994) “Optimizing locality of data representation in MLP networks,” *1994 IEEE International Conference on Neural Networks*, vol. 1, 50–4.

55. Deppisch, J., Bauer, H.-U., and Geisel, T. (1990) "Hierarchical architectures for optimized training," *International Neural Network Conference*, vol. 2, 793.
56. Stokbro, K., Umberger, D.K., and Hertz, J.A. (1990) "Exploiting neurons with localized receptive fields to learn chaos," *Complex Systems*, vol. 4, no. 6, 603–22.
57. Hakim, N.Z., Kaufman, J.J., Siffert, R.S., and Meadows, H.E. (1989) "A neural network model for prediction error identification," *Twenty-Third Asilomar Conference on Signals, Systems and Computers*, vol. 1, 119–22.
58. Grabec, I. (1992) "Prediction of chaos in non-autonomous systems by a neural network," *Proceedings on the 1992 International Conference on Artificial Neural Networks*, vol. 1, 379–82.
59. Coetzee, F., and Stonick, V. (1992) "Improving neural network performance by adapting node nonlinearities," *Proceedings of the SPIE – The International Society for Optical Engineering*, vol. 1766, 212–23.
60. Principe, J.C., Rathie, A., and Kuo, J.-M. (1992) "Prediction of chaotic time series with neural networks and the issue of dynamic modeling," *International Journal of Bifurcation and Chaos in Applied Sciences and Engineering*, vol. 2, no. 4, 989–96.
61. Rape, R., Fefer, D., and Drnovsek, J. (1994) "Time series prediction with neural networks: a case study of two examples," *1994 IEEE Instrumentation and Measurement Technology Conference*, vol. 1, 145–8.
62. Kechriotis, G.I., and Manolakos, E.S. (1993) "Using neural networks for nonlinear and chaotic signal processing," *1993 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 1, 465–8.
63. Kil, R.M., and Choi, J.Y. (1993) "Time series prediction based on fractal theory and nonlinear estimation using a network with Gaussian kernel functions," *Neural, Parallel and Scientific Computations*, vol. 1, no. 4, 477–97.



# Neuro-fuzzy systems

## Chapter outline

7.1	Introduction	337
7.2	Background	338
7.3	Architectures of neuro-fuzzy systems	339
7.4	Construction of neuro-fuzzy systems	355
7.5	Summary	359
	Problems	359
	References	361

## 7.1 Introduction

Neuro-fuzzy systems represent a newly developed class of hybrid intelligent systems combining the main features of artificial neural networks with those of fuzzy logic systems. The main goal here is to circumvent difficulties encountered in applying fuzzy logic for systems represented by numerical knowledge (data sets), or conversely in applying neural networks for systems represented by linguistic information (fuzzy sets). It is well recognized that neither fuzzy reasoning systems nor neural networks are by themselves capable of solving problems involving at the same time both linguistic and numerical knowledge.

For instance, while fuzzy logic theory permits the accurate representation of a given system behavior using a set of simple “if-then” rules, it is nevertheless unable to tackle knowledge stored in the form of numerical data. For this particular type of system, “if-then” rules have to be extracted manually from the data sets [1], a process that becomes very tedious or even impossible to achieve for data sets with large numbers of patterns. The problem becomes even harder when the knowledge about the system is stored in both forms: linguistic (fuzzy sets) and numerical (data sets). This is the case for large-scale systems usually characterized by complex dynamics and

ill-defined behavior. Neural networks on the other hand have been shown to be universal approximators capable of learning virtually any (smooth) nonlinear mapping with a high degree of accuracy. They are also known to be excellent classifiers and predictors. They accomplish this through a learning process in which numerical data are presented to the system for training under a computational structure composed of neurons and weighted links. Once a network has been trained, computation is then carried out in a parallel and distributed manner. But despite their versatility, neural networks suffer from several weaknesses among which is the implicit representation of knowledge (known among researchers as the black box structure). It is, for instance, very difficult to explicitly quantitize the meaning of weights among the nodes of the network once the systems have been trained. As such, neural networks are not very “transparent” at explaining their decision-making process [2]. In addition, it is difficult to incorporate additional knowledge into the system without retraining it. It is even more difficult to extract from the data patterns linguistic representation of knowledge.

To overcome the limitations of both system representations (fuzzy and neural), researchers in the area have proposed incorporating fuzzy logic reasoning within a learning architecture of some sort. Neural networks have been shown to be excellent candidates for this task. Automating the generation of fuzzy rules using neural networks and optimizing the parameters of the fuzzy sets have been among the major objectives of several researchers in this field for the last few years. A body of research has also tackled the issue of constructing neural networks using fuzzy neurons. While fuzzy logic and neural networks paradigms have originated from totally different mathematical formalisms, it has been shown that both of these methodologies are universal approximators for a large class of nonlinear mappings. It has also been shown that they can be combined to form a hybrid structure powerful enough to deal with a wide range of systems involving different types of knowledge (numerical and linguistic). It is the scope of this chapter to outline and highlight well-known architectures which have been developed in recent years for merging fuzzy reasoning with neural network learning.

## 7.2 Background

The integration of fuzzy logic systems with neural networks reduces the limitations of fuzzy systems in terms of lack of learning while strengthening the neural network features in terms of explicit knowledge representation. Table 7.1 summarizes the major strengths and weaknesses of both neural networks and fuzzy logic systems.

Connotations and definitions provided in the literature to describe neuro-fuzzy systems have increased dramatically over the last few years. In [3], for instance, neuro-fuzzy systems are described as hybrid systems. A number of researchers have used this term to depict systems that involve in some ways both fuzzy logic and neural network features. According to [3], there exist five classes (categories) of neuro-fuzzy systems.

**Table 7.1: Features of fuzzy logic and neural networks-based systems**

	Fuzzy logic	Neural networks
Representation	Linguistic description of knowledge	Knowledge distributed within computational units
Adaptation	Some adaptation	Adaptive
Knowledge representation	Explicit and easy to interpret	Implicit and difficult to interpret
Learning	Non-existent	Excellent tools for imparting learning
Verification	Easy and efficient	Not straightforward

- **Class A:** This class includes systems that have the same topology as classical neural networks, but with different fundamental computational elements. This means that instead of having the numerical (standard) neuron, this class employs what is known as the *fuzzy neuron*.
- **Class B:** This class includes adaptive fuzzy systems that utilize neural networks for training the fuzzy system to update the fuzzy rules or the membership functions.
- **Class C:** This class involves classical neural networks, for which the learning is achieved through fuzzy systems-based functions, that is, by using fuzzy methods to update the weights of the neurons.
- **Class D:** Systems in this class are created by using a structure involving independently working fuzzy systems and neural network modules.
- **Class E:** This class is a mixture of the classical systems and any other system from the above classes.

Although it is intuitive to classify neuro-fuzzy systems as has been done above, it is nevertheless difficult to highlight the major differences that exist among each of them. This is due to the fact that the integration of fuzzy logic and neural networks produces systems with different features, structures, and types of application. More specifically, the term neuro-fuzzy is used to depict systems that incorporate neural methods into a fuzzy logic inference structure. The aim of this integration is to either adapt fuzzy systems to the changing environment, or to identify fuzzy rules and membership functions. In the next section, we provide another way for classifying neuro-fuzzy systems according to architectures. This classification is regarded today among the most accepted ones in the field.

## 7.3 Architectures of neuro-fuzzy systems

Neuro-fuzzy systems (NFS) can be divided into three major types according to their topologies and functionalities ([4] and [5]). They are the *cooperative neuro-fuzzy systems*, the *neural network-driven fuzzy reasoning systems* and

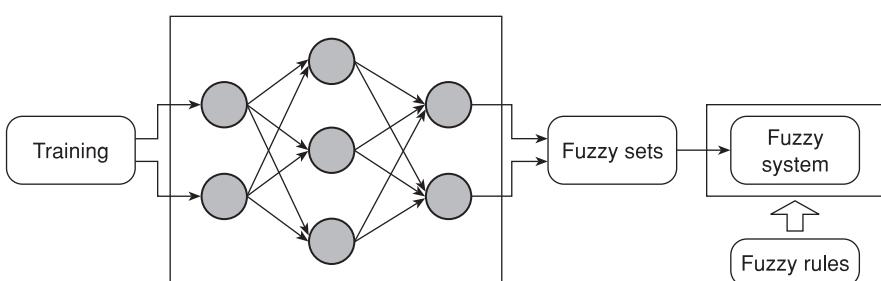
the *hybrid neural network-based systems*. These systems are described in more detail next.

### 7.3.1 Cooperative neuro-fuzzy systems

This class of neuro-fuzzy systems, also known as *cooperative neuro-fuzzy systems* [5], has as its main task the optimization and tuning of the neural networks involved in the architecture. The idea is to optimize and tune the fuzzy inferencing structure using some sort of learning mechanism. This type of NFS structure lacks parallelism, in that both the neural network and the fuzzy system operate virtually independently of each other. As such, while the learning takes place in the neural network at a certain level, the fuzzy inference system remains unchanged during this stage.

In *cooperative neuro-fuzzy systems*, the combination of neural networks and fuzzy inferencing aims at optimizing the structure of a given fuzzy system through learning provided by a neural network. This can be achieved by utilizing a neural network to determine certain parameters of the fuzzy system, a process that can be carried out either off line or on line. According to the parameter type of the fuzzy system, cooperative neuro-fuzzy systems can be classified into four different kinds as shown in [5], each one utilizing neural networks in a particular configuration.

- (1) A set of training data is used here to determine the membership functions for a fuzzy system. This is done through a neural network structure, the output of which is a number of fuzzy sets. This is later combined with a predetermined set of fuzzy rules to implement the structure of the fuzzy system. Figure 7.1 illustrates the schematic representation of such a *cooperative neuro-fuzzy system*.
- (2) Through a data clustering approach, the neural network structure in this scheme is used to identify or extract the fuzzy rules from the training data. The neural network learns the rules before implementing the fuzzy system, which receives predetermined information about the fuzzy sets and the structure of their membership functions. Figure 7.2 illustrates this type of *cooperative neuro-fuzzy system*.



**Figure 7.1:** Cooperative neuro-fuzzy type 1

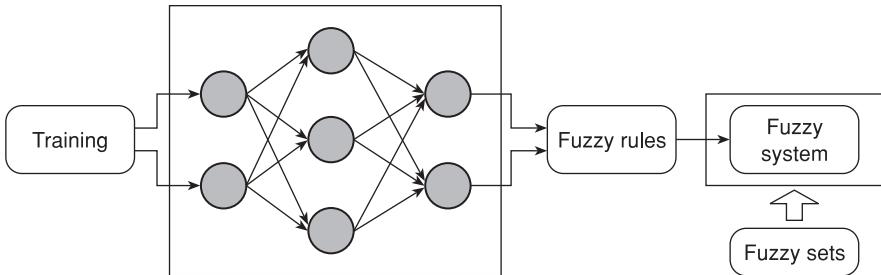


Figure 7.2: Cooperative neuro-fuzzy type 2

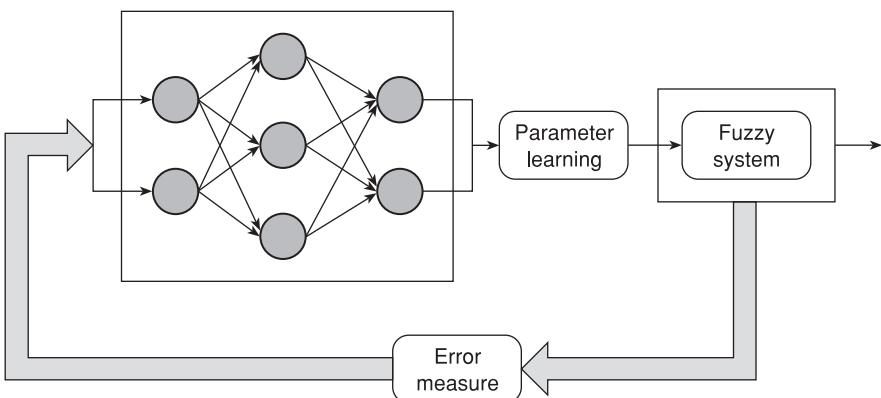
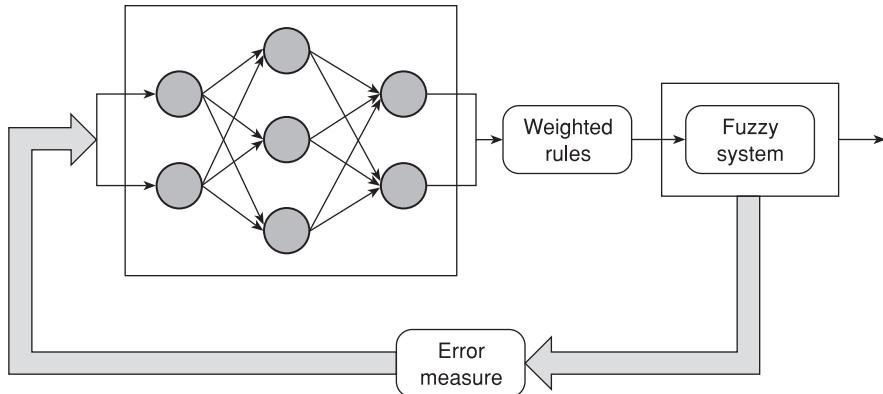


Figure 7.3: Cooperative neuro-fuzzy type 3

- (3) The neural network is used here to regularly update the fuzzy system structure. The adaptation is applied to the membership functions during the on-line use of the fuzzy system. In this approach, initial fuzzy rules and membership functions must be specified, and in addition, a feedback from the fuzzy system (error measure) should be specified to guide the learning process. A schematic representation of such a system is shown in Figure 7.3.
- (4) In this structure, a learning approach provides means for setting the importance of each rule in the fuzzy system. Weights are then assigned for each of these rules using a similar procedure to that provided in [7]. In [8], Blanco *et al.* introduced an interesting approach to identify the weighted rules using a neural network structure. They proposed to choose the models to train the network according to the number of crisp rules in the system, or by creating a continuous system dependent on a continuous initial system. This type of *cooperative neuro-fuzzy structure* is illustrated in Figure 7.4.

#### 7.3.1.1 Neural networks for determining membership functions

The use of neural networks to determine the membership functions is considered as a data clustering and classification approach [9]. Here, one can apply a



**Figure 7.4:** Cooperative neuro-fuzzy type 4

neural network-based algorithm for clustering and classification to solve the problem. Adeli and Hung [10] introduced an algorithm for determining the membership function. They stated that the use of neural networks for producing the membership function involved two stages: clustering and fuzzification. In the first stage a neural network is used to cluster or classify data into certain number of clusters. In the fuzzification process (stage) the fuzzy membership values are assigned for each training stage.

#### 7.3.1.2 Adeli–Hung algorithm (AHA)

The AHA algorithm assumes that the data consists of  $N$  training instances  $X_1, X_2 \dots X_N$ , each with  $M$  patterns  $\{x_{i1}, x_{i2}, \dots, x_{im}\}$ . The mean vector of the instances can be defined as

$$\bar{X}_N = \frac{1}{N} \sum_{i=1}^N X_i \quad (7.1)$$

For  $N + 1$  instances, the mean vector is derived from the mean  $\bar{X}_N$  and is expressed as:

$$\bar{X}_{N+1} = \frac{1}{N+1} \sum_{i=1}^{N+1} X_i = \frac{N}{N+1} \bar{X}_N + \frac{1}{N+1} X_{N+1} \quad (7.2)$$

In AHA the classification is performed using a topology-and-weighted change, a neural network with  $M$  inputs equal to the number of patterns ( $M$ ) and a number of output nodes equal to the number of clusters. The following procedure for classifying training data instances into an active or new cluster is used in AHA:

**Step 1:** Calculate the degree of difference  $diff(X, C_i)$  between the instance  $X$  and each cluster  $C_i$ . This can be achieved using a Euclidean distance and the function  $diff(X, C_i)$  given as

$$diff(X, C_i) = \sqrt{\sum_{j=1}^M (x_j - c_{ij})^2} \quad (7.3)$$

*Step 2:* Define the active clusters. The active cluster is the cluster with the smallest degree of difference.

$$C_{active} = \{C, \min\{diff(X, C_i), i = 1, 2, \dots, P\}\} \quad (7.4)$$

*Step 3:* Define a cluster as new if the  $diff(X, C_i)$  is greater than a predefined threshold value,  $k$ .

$$C_{new} = X \quad \text{if } k < \min\{diff(X, C_i), i = 1, \dots, P\} \quad (7.5)$$

Now suppose that the  $N$  training instances have been classified into  $P$  clusters. A prototype for each cluster is defined as the mean of all instances in that cluster:

$$C_p = [c_{p_1}, c_{p_2}, \dots, c_{p_M}] = \frac{1}{n_p} \sum_{i=1}^{n_p} X_i^p \quad (7.6)$$

The degree of membership for each instance in the cluster is based on the similarity of this instance to its prototype. The similarity can be defined as the distance function  $D^w$  between the instance and the prototype given as:

$$D^w(X_i^p, C_p) = \|w_p(X_i^p, C_p)\|^w = \sqrt{\sum_{j=1}^M (x_{ij}^p - c_{pj})^2}. \quad (7.7)$$

The fuzzy membership value of the  $j$ -th instance in the  $P$  cluster can be defined – using triangular-shaped membership functions over the  $diff$  universe – as:

$$\mu_p(X_i^p) = f[D^w(X_i^p, C_p)] = \begin{cases} 0 & \text{if } D^w(X_i^p, C_p) > k \\ 1 - \frac{D^w(X_i^p, C_p)}{k} & \text{if } D^w(X_i^p, C_p) \leq k. \end{cases} \quad (7.8)$$

### 7.3.1.3 Learning fuzzy rules using neural networks

In recent years, research work has tackled the issue of learning and automatic extraction of fuzzy rules [11], [12], [6], [13], and [5]. Neural networks play a key role in inducing the fuzzy rules from input–output training space. Using the input–output partitioning techniques (or clustering approaches), the neural network can extract and learn about fuzzy rules. Usually, a self-organization map [14] or a similar architecture is used here. The membership functions should be defined separately [5]. One method for extracting fuzzy rules was proposed by Pedrycz and Card in [12] and was based on the linguistic interpretation of self-organizing feature maps.

### 7.3.1.4 Learning in fuzzy systems using neural networks

A neural network can be used to learn and adapt the parameters (the membership function shapes) of a fuzzy logic system. An error signal must be provided to the neural network as a feedback to guide the learning process. In general, most fuzzy system adaptation schemes utilize the gradient-descent optimization. In such schemes the main purpose of learning is to minimize an objective function  $E$  [9] given by:

$$E = \frac{1}{2}(t - o)^2, \quad (7.9)$$

where  $t$  is the desired output and  $o$  is the actual output.

### 7.3.1.5 Identifying weighted fuzzy rules using neural networks

In structure identification (using input–output data) of fuzzy systems, the weights of the fuzzy rules can reflect the consistence level (accuracy) of the rules. The weights are multiplied by the rule outputs and they can affect the interpretation of the linguistic variables in different rules. Blanco *et al.* [8] presented a learning procedure to identify the weights of the rules using neural networks.

## 7.3.2 Neural network-driven fuzzy reasoning

For a given fuzzy system, if the number of linguistic variables is four or more, it may become difficult to construct the fuzzy rule base [15]. Many researchers have discussed the problem of automating the extraction of the fuzzy rules [16], [17], [18], and [19]. One important scheme is the *neural network-driven fuzzy reasoning* (NNDF) introduced by Takagi and Hayashi [6]. This scheme is proposed to solve problems pertaining to (a) systems where techniques other than heuristic approaches are needed to design the membership function; and (b) systems where reasoning must be adaptable to changes in the environment. The basic idea of the Takagi–Hayashi method is to implement the membership functions in the antecedent in the inference functions of the consequent. To implement this scheme, a suitable neural network is used. The Takagi–Hayashi method consists of three major parts, as shown in Figure 7.5. The parts are:

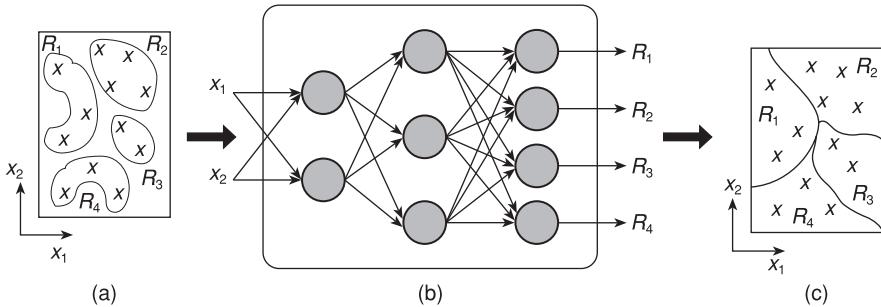
*Part 1:* Partitioning the inference rules.

*Part 2:* Identifying a given rule's If (LHS) part (determines the membership function).

*Part 3:* Identifying a given rule's Then (RHS) part.

The Takagi–Hayashi method makes use of neural networks to generalize the Sugeno-type fuzzy inference system. Hence, the rules in this method take the following format:

$$R^p: \text{If } X = (x_1, \dots, x_n) \text{ is } A^p \text{ then } O_p = NN_p(x_1, \dots, x_m) \quad p = 1, \dots, s \quad (7.10)$$



**Figure 7.5:** Neural network-driven fuzzy reasoning

where  $s$  is the number of inference rules which represent a fuzzy set for the antecedent part of each inference rule, and  $NN_p(\cdot)$  is a structure of a model function characterized by a backpropagation learning-based neural network similar to what was defined in [20].

The design procedure for a Takagi–Hayashi system consists of the following steps:

*Step 1* Select the input–output variables and the training data. Let  $y_i$  be the output and  $x_j$  be the input variables with  $j = 1, 2, \dots, k$ .

*Step 2* Divide the input–output data  $(x_i, o_i)$  into two sets of data; training data of size  $n_t$  and checking data of size  $n_c$ , where  $n = n_t + n_c$ .

*Step 3* Using a clustering method, find the partition of the training data.

*Step 4* Identify the *if*-part of each fuzzy rule. This can be achieved by training a neural network to generate the membership functions.

*Step 5* Train  $NN_p$  corresponding to the *then*-part of fuzzy inference. Here, the input–output relation determines the structure of the *then*-part for each fuzzy rule.

*Step 6* Simplify the *then*-parts with a backward elimination method.

*Step 7* Make a decision on the final output.

### 7.3.3 Hybrid neuro-fuzzy systems

*Hybrid neuro-fuzzy systems* are based on an architecture which integrates in an appropriate parallel structure a neural network and a fuzzy logic-based system. Unlike cooperative neuro-fuzzy systems, the fuzzy logic system and the neural network work as one synchronized entity.

#### 7.3.3.1 Architecture of hybrid neuro-fuzzy systems

*Hybrid neuro-fuzzy systems* have a parallel architecture, and exploit similar learning paradigms, as is the case for neural networks. The parallelism of the system can be viewed as a mapping of the fuzzy system into a neural network

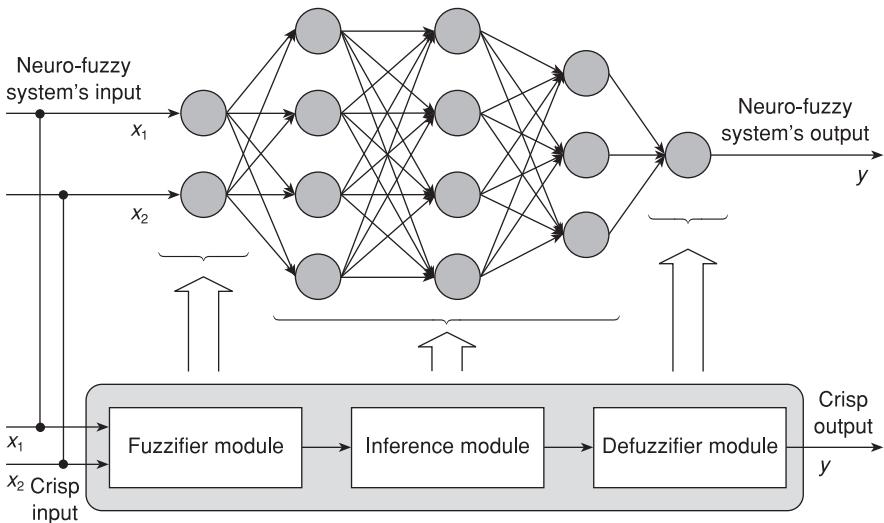


Figure 7.6: Hybrid neuro-fuzzy systems

structure. In other words, each functional module of the fuzzy logic system corresponds to a particular layer of the neural network. The resulting system can be interpreted either as a neural network or a fuzzy logic system. Figure 7.6 depicts the mapping from a fuzzy logic system to a neural network structure.

Hybrid neuro-fuzzy systems have the same architecture as that of traditional fuzzy systems except that a layer of hidden neurons performs each of the tasks of the fuzzy inference system. As such, a fuzzy logic inference system can be implemented as a five-layer neural network. This type of architecture is the most commonly used among neuro-fuzzy inference systems, and it uses the Sugeno-type fuzzy inferencing. The structure of the system can be described layer by layer as follows:

#### *Layer 1. Fuzzification*

This layer consists of a set of linguistic variables. The crisp inputs  $x_1$  and  $x_2$  are fuzzified through mapping into membership functions  $A_i$  and  $B_i$  of the linguistic variables, which usually take triangular, trapezoid, or bell-shaped membership functions.

#### *Layer 2. Rule nodes*

The second layer contains one node per each fuzzy *if-then* rule. Each rule node performs a connective operation within the rule antecedent (*if* part). Usually, the minimum or the dot product is used as the T-norm operator to represent the connective AND. The union OR connective is usually represented using the maximum operation or any other T-conorm operator.

#### *Layer 3. Normalization*

In this layer, the firing strengths of the fuzzy rules are normalized. The normalization is to calculate the ratio of the  $p$ -th rule's firing strength to the sum of all rules' firing strengths:

$$\bar{w}_p = \frac{w_p}{\sum_p w_p} \quad (7.11)$$

where  $w_p$  is the firing strength of the  $p$ -th rule.

#### *Layer 4. Consequent layer*

This layer is related to the consequent of the fuzzy rule. The values of the consequent (*then* part) are multiplied by normalized firing strengths according to:

$$\bar{o}_p = \bar{w}_p o_p \quad (7.12)$$

#### *Layer 5. Summation*

This layer computes the overall output as the summation of the incoming signals:

$$o^* = \sum_p \bar{o}_p \quad (7.13)$$

Although the above architecture of the hybrid neuro-fuzzy system is the most commonly known, it is not the only one involving mapping of fuzzy systems into neural networks. This mapping can be achieved using more or fewer layers. In [21], an on line self-constructing neuro-fuzzy inference system was proposed using six layers. L. Maguire *et al.* presented a neuro-fuzzy system to approximate fuzzy reasoning using only three layers [22]. A common example of a hybrid neuro-fuzzy system is the *Adaptive-Network-based Fuzzy Inference System* (ANFIS), which can be represented in a five- or four-layer architecture. This structure is described next.

#### 7.3.3.2 Five-layer neuro-fuzzy systems

An example of the five- and four-layer hybrid neuro-fuzzy system is the *Adaptive-Network-based Fuzzy Inference System*. It was proposed by J-S Jang [23]. ANFIS represents a Sugeno-type fuzzy system, where the fuzzy rules take the following form:

$$R_p: \text{If } x_1 \text{ is } A_1^p \text{ and } x_2 \text{ is } A_2^p \dots \text{ and } x_n \text{ is } A_n^p \text{ Then } o_p = \alpha_0^p + \alpha_1^p x_1 + \dots + \alpha_n^p x_n \quad (7.14)$$

where  $x_i$  is  $i$ -th input linguistic variable in the antecedent part of the  $p$ -th rule with  $i = 1, \dots, n$  and  $A_i^p$  is the linguistic label associated with it in that rule.  $A_i^p$  has its associated fuzzy membership function given by  $\mu_{A_i^p}(x_i)$ .  $o_p$  is the consequent output of the  $p$ -th rule and  $\alpha_0^p, \dots, \alpha_n^p$  are the Sugeno parameters.

The structure of ANFIS consists of a five-layer feedforward network. The nodes in each layer have the same functionality. ANFIS only supports Sugeno-type systems, with the following constraints:

- First or zero order Sugeno-type systems.
- Single output, obtained using a weighted average defuzzification method (linear or constant output membership functions).
- The weight of each rule is unity.

Figure 7.7 illustrates the ANFIS five-layer architecture for a two-input ( $i = 1, 2$ ) one-output first order Sugeno fuzzy model with four rules ( $p = 4$ ).

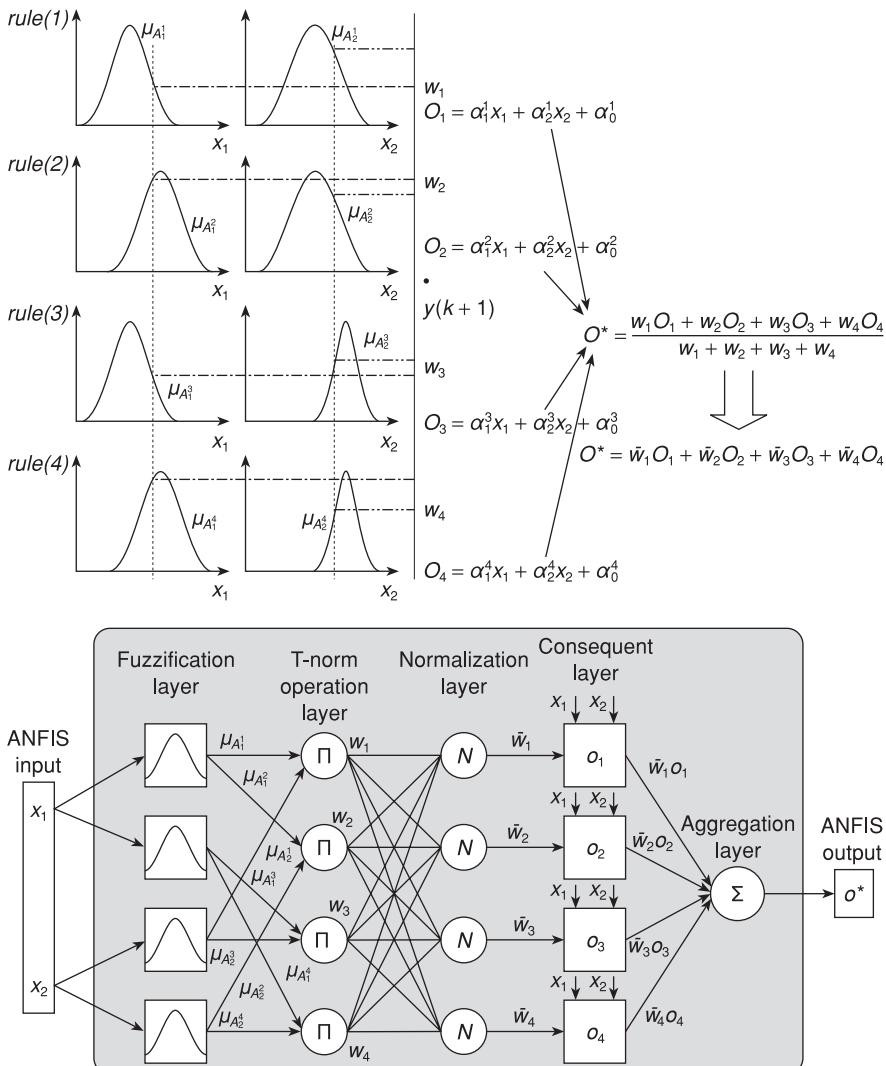


Figure 7.7: Five-layer ANFIS

**Layer 1:**

Every node in this layer is characterized by its corresponding output function:

$$O_i^1 = \mu_{A_i^p}(x), \quad (7.15)$$

where  $x$  is the input to the node, and  $A_i^p$  is the linguistic label associated with this node function. Each node in this layer stores three parameters  $a_p$ ,  $b_p$ ,  $c_p$  to define a bell-shaped membership function that represents the profile of the linguistic variable  $x_i$ .

$$\mu_{A_i^p}(x_i) = \frac{1}{1 + \left( \left( \frac{x_i - c_p}{a_p} \right)^2 \right)^{b_p}} \quad (7.16)$$

**Layer 2:**

This layer stores the rules. Each rule is represented by one node. Each node is connected to those nodes in the previous layer that forms the antecedent of the rule. For a node  $r$ , the inputs are degrees of membership functions, which are multiplied through a T-norm operator  $\otimes$  to determine the degree of fulfillment  $w_p$  of the rule.

$$w_p = \mu_{A_i^p}(x_1) \otimes \mu_{A_j^p}(x_2), \quad i = 1, 2; j = 1, 2; p = 1, \dots, 4 \quad (7.17)$$

**Layer 3:**

In this layer a unit computes the relative degree of fulfillment for every rule  $R_p$ :

$$\bar{w}_p = \frac{w_p}{\sum_{p=1}^4 w_p} \quad (7.18)$$

**Layer 4:**

The nodes of this layer are connected to all input nodes and with exactly one node in layer three. Each node in this layer computes the output for the rule. In other words, it computes the consequent of the rule (*then* part):

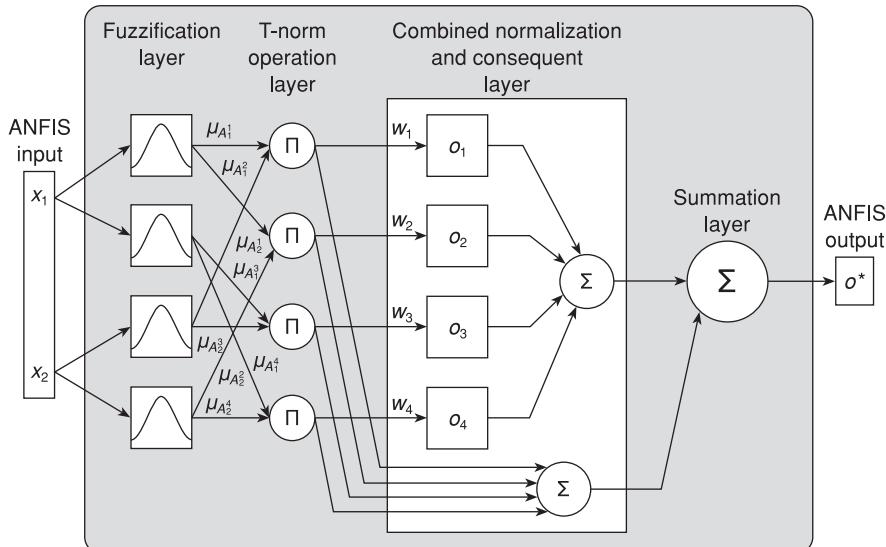
$$O_p^4 = \bar{w}_p o_p = \bar{w}_p (\alpha_0^p + \alpha_1^p x_1 + \alpha_2^p x_2) \quad (7.19)$$

where  $\bar{w}_i$  is the output of layer 3, and  $\{p_i, q_i, r_i\}$  is the parameter set.

**Layer 5:**

An output node computes the final output as the summation of all incoming signals from layer 4.

$$O^5 = \sum_{p=1}^4 \bar{w}_p (\alpha_0^p + \alpha_1^p x_1 + \alpha_2^p x_2) \quad (7.20)$$



**Figure 7.8:** Four-layer ANFIS

### 7.3.3.3 Four-layer neuro-fuzzy systems (ANFIS)

Although the five-layer structure described here has been so far the most commonly known for ANFIS architecture, there has been another suggested structure of ANFIS involving a four-layer architecture. In this structure, layers three and four pertaining to normalization and consequent variables are combined in such a way as to obtain an equivalent structure with only four layers. In the new network the weight normalization takes place at the last layer [15]. Figure 7.8 shows the ANFIS four-layer structure with two inputs.

### 7.3.3.4 Three-layer neuro-fuzzy approximator

In [22], L. Maguire and co-authors introduced a three-layer neuro-fuzzy system as an alternative architecture for approximate fuzzy reasoning. The fuzzy model used in their work is a zero-order Sugeno model, where each fuzzy rule has a fuzzy singleton consequent. The proposed neuro-fuzzy architecture implements the zero-order Sugeno type as a three-layer network. The first layer consists of  $n + 1$  nodes where  $n$  is the number of inputs. The extra node represents a unity bias input. The input layer (first layer) broadcasts the inputs to selected nodes in the hidden layer. The number of nodes in the hidden layer depends on the number of fuzzy sets associated with each input domain. The number of nodes in the hidden layer is determined by the relation  $np$ , where  $p$  is the number of fuzzy sets (partitions) in each input domain and  $n$  is the number of inputs.

Each node in the hidden layer receives two weighted inputs, one from one of the input variables and the second from the bias node. In the hidden node, the squared sum of those variables is passed to an activation function to produce an output according to:

$$o_i = \exp\{-(\sum w_{ij}x_j + w_i - 1)^2\}, \quad (7.21)$$

where  $o_i$  is the output of the  $i$ -th node in the hidden layer,  $w_{ij}$  the weight of the connection between the  $i$ -th node in the hidden layer and the  $j$ -th node in the input layer and  $w_i$  is the weight of connection between the  $i$ -th node in the hidden layer and the bias node in the input layer.

Since the number of nodes in the hidden layer is equal to the number of fuzzy sets, membership values can be used to represent rules' weights ( $\beta$ ) in the fuzzy reasoning process. The output layer consists of one node that represents the weighted sum of  $o_i$

$$O^* = \sum_{i=1}^{np} \beta_i o_i \quad (7.22)$$

where  $\beta_i$  is the weight of connection between the  $i$ -th node in the hidden layer and the output layer.

Three benchmark applications were often used to evaluate the performance of the neuro-fuzzy system, particularly the ANFIS architecture:

- (a) Nonlinear function approximation.
- (b) On line identification in control system.
- (c) Chaotic time-series prediction.

In most cases, it has been proved that the proposed neuro-fuzzy architecture has a better performance with respect to the following aspects:

- (a) Implementation accuracy.
- (b) Training time.
- (c) Network dimension (size).

Some of these aspects are demonstrated in the next example pertaining to a 3-D nonlinear function approximation.

### Example 7.1

For illustration purposes, we present next some results demonstrating the powerful approximation capabilities of ANFIS and compare them with those of a conventional BPL-based neural network. The example pertains to approximating the 3-D surface of the well-known Peaks function, which is given by the expression:

$$g(x, y) = 3(1-x)^2\exp(-x^2 - (y+1)^2) - 10(x/5 - x^3 - y^5)\exp(-x^2 - y^2) - (1/3)\exp(-(x+1)^2 - y^2) \quad (7.23)$$

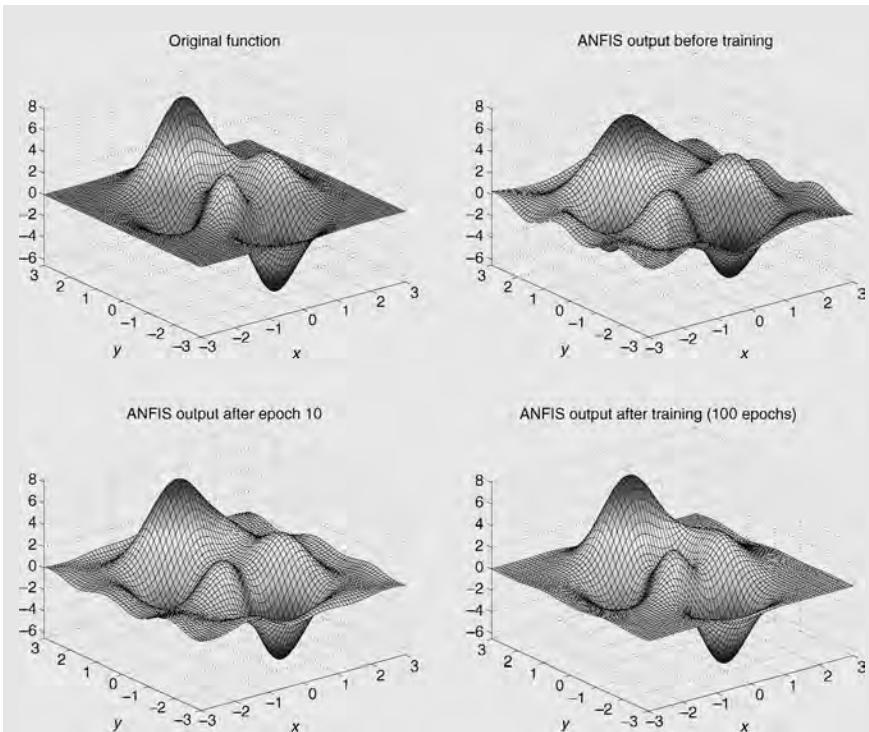


Figure 7.9: The Peaks function (original function and its ANFIS approximation)

The range of approximation is chosen here as  $(x, y) \in [-2, 2] \times [-2, 2]$ . Figure 7.9 shows the original function and the approximated ones obtained by ANFIS before training, after 10 epochs, and after 100 epochs of training.

The corresponding membership functions for the input variables  $x$  and  $y$  are also shown in Figure 7.10 before training, after 10 epochs of training, and after 100 epochs of training. In this figure one could clearly notice the difference in terms of the membership function shapes and their spacing with respect to each other.

Figure 7.11 on the other hand shows the error surface obtained before training, after 10 epochs, and after 100 epochs of training, respectively. The accuracy of the results once the system has been trained enough is clearly shown in this figure.

Figure 7.12 quantifies the accuracy of ANFIS when compared to a BPL-based network trained with 300 sets of input/output data and using a structure with two inputs, six hidden nodes located in a single hidden layer and one output. The figure also shows the asymptotic decay behavior of the error in terms of the number of epochs for the ANFIS case and for the BPL case.

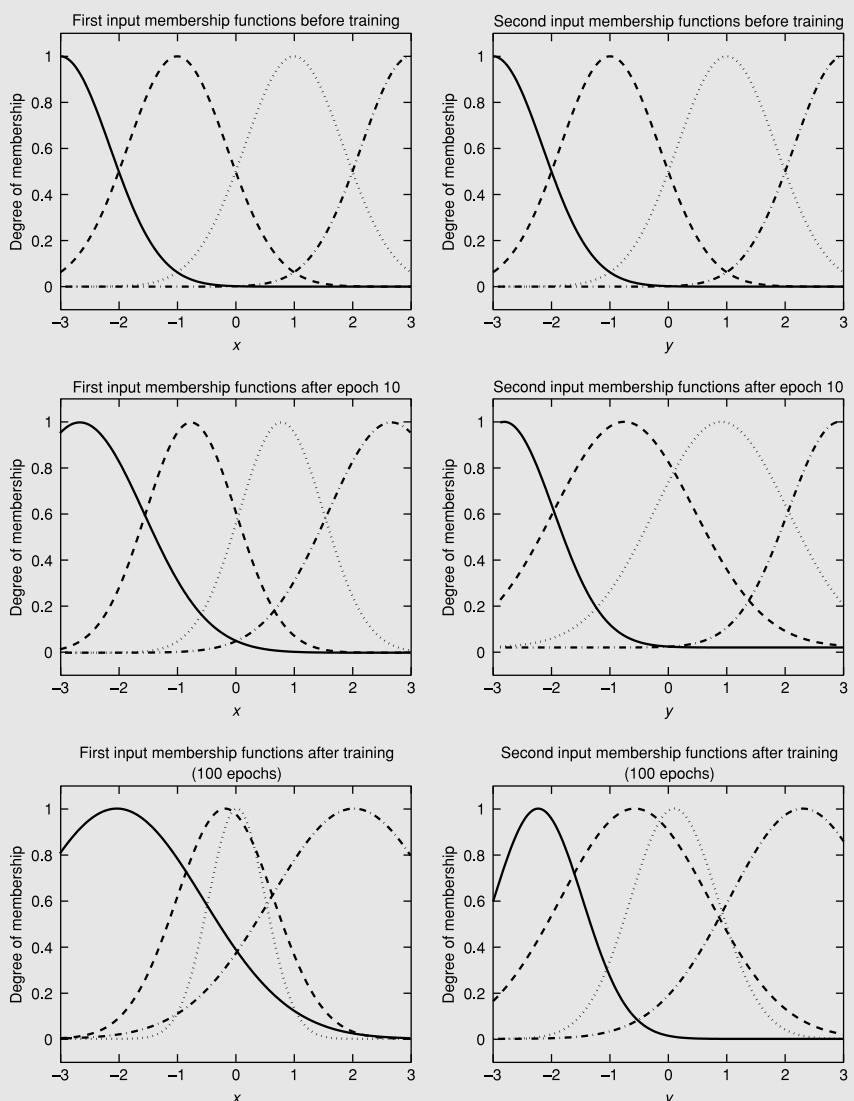
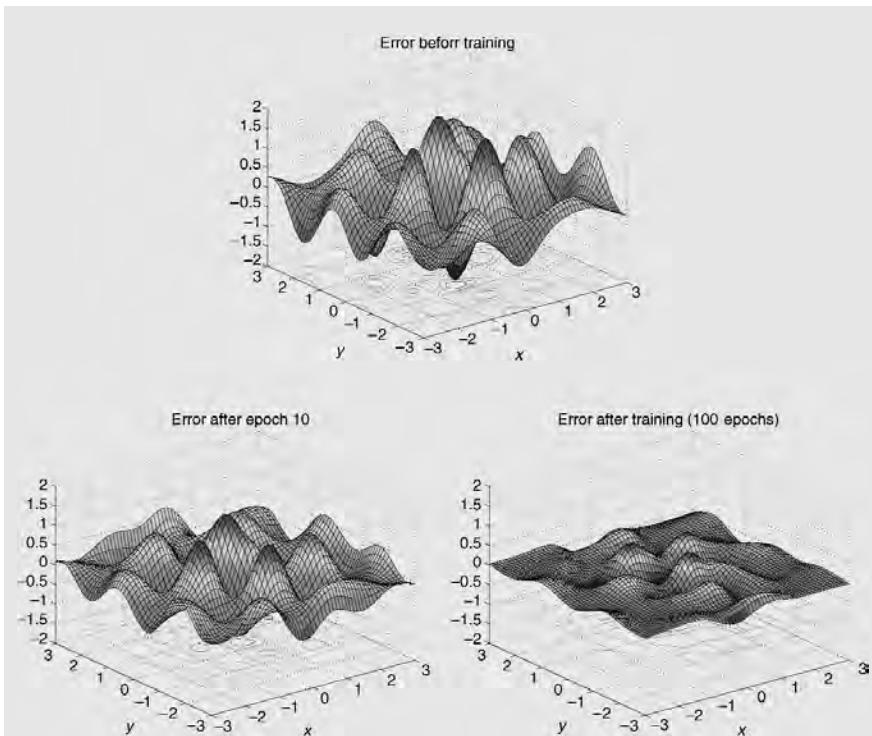
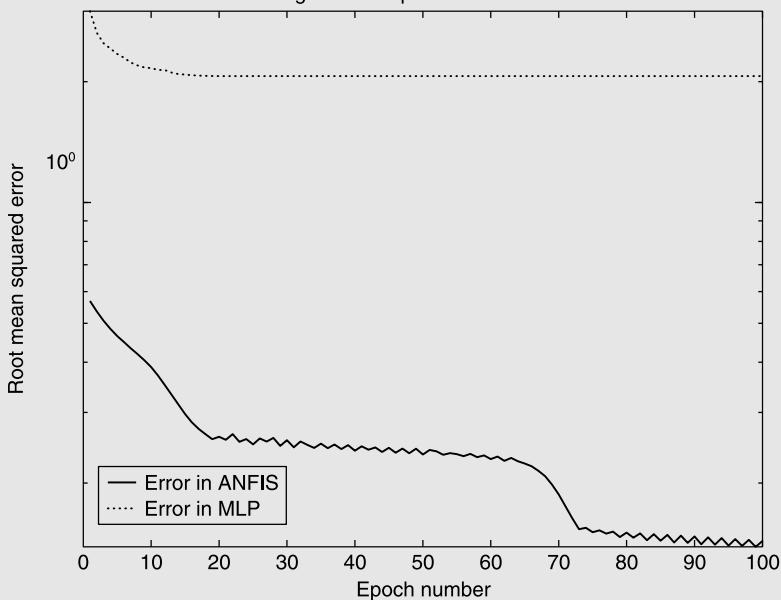


Figure 7.10: Evolution of the ANFIS membership functions

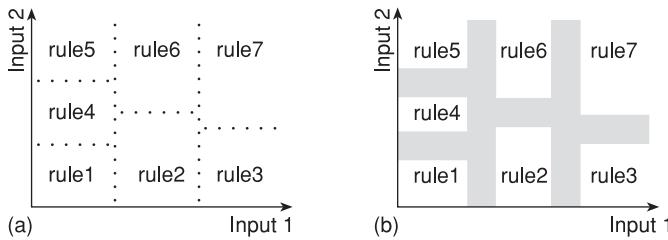


**Figure 7.11:** ANFIS output error quantification

Training error comparison: ANFIS vs MLP



**Figure 7.12:** Performance of ANFIS versus MLP



**Figure 7.13:** Crisp and fuzzy partitioning

## 7.4 Construction of neuro-fuzzy systems

Usually, to construct a neuro-fuzzy system we use a set of numerical data consisting of an input–output space. The construction of the system involves two essential phases: the structure learning phase (structure identification), which aims to determine the fuzzy rules structure, and the parameter learning phase used to tune and optimize the parameters of each fuzzy rule constructed in the first phase. These two phases are carried out in sequence with the learning structure phase coming first followed by the learning parameter phase. These two phases are discussed in the following sections.

### 7.4.1 Structure identification phase

The structure identification is based on the partitioning of the input–output space. After the partitioning process, each partition represents one rule. In systems that utilize crisp rules, there is no overlap between the boundaries of partitions. However, this overlap exists in fuzzy systems. Figure 7.13 explains the difference between the partitions of (a) fuzzy and (b) crisp rules. The input–output space partitioning is a technique used to form the antecedent of fuzzy rules and explain how the fuzzy rules are related to the input space. The structure and the number of fuzzy rules are influenced by the partitioning techniques.

There are several partitioning methods. The most common ones are given below.

#### 7.4.1.1 Grid-type partitioning

During the phase of structure identification of the fuzzy inference system (antecedent), fuzzy partitioning techniques are used. Here, the multidimensional population of input–output data is divided into several partitions. Following that, each partition is made to correspond to one single fuzzy rule. Figure 7.14 shows the common fuzzy grid partitioning, where Figure 7.14(a) illustrates the “fixed” grid partitioning, while Figure 7.14(b) shows the adaptive grid partitioning. In the first type, the multidimensional input–output data space is divided into “fixed” partitions, in which no adaptation is made to the partitions. On the other hand, the fuzzy adaptive grid-partitioning applies adaptation for the partitions. In other words, there is no formal way

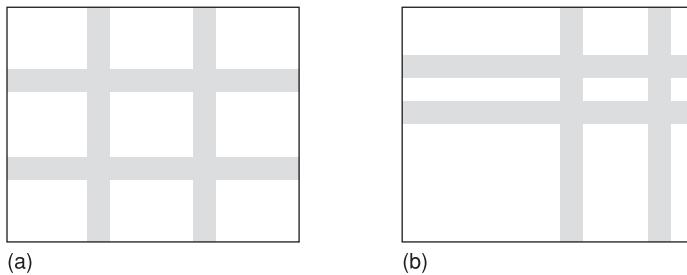


Figure 7.14: Fixed (a) and adaptive (b) grid partitioning

to adapt the antecedent of fuzzy rules in fixed grid partitioning. This adaptation method can be found in the fuzzy adaptive grid-partitioning approach.

While the grid-partitioning approach has the advantage that all fuzzy rules can be generated using a relatively small number of linguistic variables, it suffers nevertheless from serious disadvantages. The size of fuzzy rules generated by the grid-partitioning techniques grows exponentially.

This problem is known as the *curse of dimensionality*. To explain this problem, suppose we have a system with  $n$  inputs and  $k$  membership functions for each input, then the number of fuzzy rules generated is equal to  $k^n$ . For example a system with 10 inputs and three membership functions associated with each input will have  $3^{10} = 59049$  fuzzy rules using fixed grid partitioning. For more information on this and other issues related to this technique, one may refer to [15] and [24].

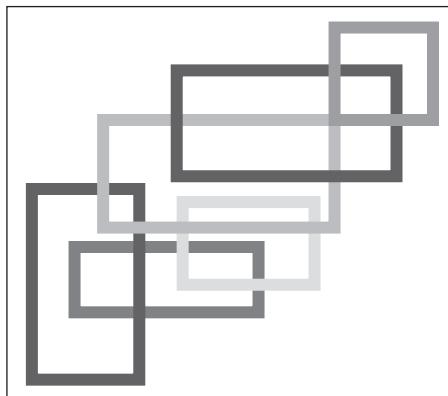
#### 7.4.1.2 Clustering

The idea of clustering is based on how to partition a collection of objects (vector of data points) into distinct subsets or clusters. The objects belonging to the same cluster have the same properties. In “hard” clustering techniques, each object (data point) is assigned to one and only one cluster, without any degree of membership of other clusters. However, in fuzzy clustering techniques, each data point belongs to a cluster with a specific degree determined by a membership function value. Hence, a data point could belong to more than one cluster with different membership values. The most popular technique is the fuzzy  $c$ -mean clustering technique.

The fuzzy  $c$ -mean is a clustering technique introduced by Dunn and extended by Bezdek [25]. The fuzzy  $c$ -mean algorithm is an extension of the classical  $c$ -mean algorithm where the data points are divided into clusters according to the distances between each point and what is called the “center of clusters.”

To understand the  $c$ -mean algorithm and the fuzzy  $c$ -mean algorithm, let  $X = \{x_1, x_2, \dots, x_n\} \subset R^d$  be a set of  $n$  vectors representing the data population. The location of the cluster  $i$  ( $i$ -th cluster center), for  $m \geq 1$ , can be defined as a vector  $v$ :

$$v_i = \sum_k (\mu_{ik})^m x_k / \sum_k (\mu_{ik})^m, \quad (7.24)$$



**Figure 7.15:** Scatter partitioning

where  $k$  is an object (data points) in  $x_i$  and  $\mu_{ik}$  is the membership (which is 1 or 0 in the  $c$ -mean algorithm) of the  $k$ -th object. In the classical  $c$ -mean, for a given integer  $c \geq 2$ , the subsets  $s_1, \dots, s_c$  of  $x$  are chosen to minimize

$$\sum_i \sum_{x \in S_i} \|x - v_i\|^2, \quad (7.25)$$

where  $v_i \in R^d$  and  $\|x\|$  denotes the Euclidean norm of the vector ( $x = (\sum_j x_j^2)^{1/2}$ ). The fuzzy  $c$ -mean algorithm produces fuzzy clustering in a similar way. For  $c \geq 2$  and  $m \geq 1$ , the algorithm chooses  $\mu_i: x \rightarrow [0, 1]$ , so that  $\sum_i \mu_i = 1$ , and  $v_i \in R^d$  for  $i = 1, 2, \dots, c$ , to minimize the function  $\sum_i \sum_k (\mu_{ik})^m \|x_k - v_i\|^2$ . In order to minimize the objective function, the values of membership functions are chosen so that the high values occur for data points or objects close to the corresponding cluster centers.

#### 7.4.1.3 Scatter partitioning

One approach to partitioning the multidimensional input–output space into rule patches has been proposed to limit the number of fuzzy rules generated. This approach is known as scatter partitioning. The basic idea behind scatter partitioning is to allow the antecedent (*if*-part) of the fuzzy rules to be positioned at arbitrary locations in the input space. However, a problem arises in scatter partitioning and it pertains to finding a suitable number of rules and suitable positions and width of the rule patches in the input space [24]. Figure 7.15 illustrates the scatter-partitioning concept.

#### 7.4.2 Parameter learning phase

As mentioned above, the learning of the neuro-fuzzy system consists of two main parts: the structure learning phase and parameter learning phase. The latter deals with the modifying and adjustment of systems parameters such as membership functions and the connection weights. In this section we discuss the most common learning algorithms that are used in neuro-fuzzy systems.

**Table 7.2:** Examples of learning algorithms

Unsupervised learning	Supervised learning
<p><b>Recurrent Networks</b></p> <ol style="list-style-type: none"> <li>1. Adaptive resonance theory (ART1)</li> <li>2. Analog adaptive resonance Theory (ART2, ART2a)</li> <li>3. Temporal associative memory (TAM)</li> <li>4. Kohonen self-organizing map (SOM)</li> <li>5. Competitive learning</li> </ol> <p><b>Feedforward Networks</b></p> <ol style="list-style-type: none"> <li>1. Fuzzy associative memory (FAM)</li> <li>2. Linear associative memory (LAM)</li> <li>3. Learning matrix (LM)</li> <li>4. Driver-reinforcement learning (DR)</li> <li>5. Optimal linear associative Memory (OLAM)</li> </ol>	<p><b>Recurrent Networks</b></p> <ol style="list-style-type: none"> <li>1. Fuzzy cognitive map (FCM)</li> <li>2. Backpropagation through time (BPTT)</li> <li>3. Real-time recurrent learning (RTRL)</li> </ol> <p><b>Feedforward Networks</b></p> <ol style="list-style-type: none"> <li>1. Perceptron learning rule</li> <li>2. Backpropagation (BP)</li> <li>3. ARTMAP</li> <li>4. Learning vector quantization (LVQ)</li> <li>5. Orthogonal least square (OSL)</li> </ol>

The *learning algorithms* are the strategies for optimizing the weights and the biases of the network. As a matter of fact, learning rules highly influence the efficiency of the system. Learning algorithms are usually classified into three major categories: *supervised learning*, *unsupervised learning*, and *reinforcement learning*. An excellent discussion of the different training algorithms can be found in [20] and [26] and a good overview was given in Chapter 4.

During the 1970s, as mentioned previously, neural network research went through a latent period due to the lack of powerful learning algorithms. This was the case until the introduction of the backpropagation training algorithm by Rumelhart. Ever since, many learning algorithms have been introduced to enhance the performance and the efficiency of the learning procedure. These learning algorithms can be categorized according to the learning type or the network topology (feedforward or recurrent). However, the main categorization of these learning methods is based on the learning type: supervised or unsupervised learning. Table 7.2 illustrates some examples of the learning algorithms used in neuro-fuzzy research.

#### 7.4.2.1 The backpropagation learning algorithm

The BPL has been discussed in detail in Chapter 4. The BPL training algorithm is a powerful learning procedure, which led to the development of many other powerful learning algorithms used today in several applications. Despite its many advantages, it still suffers from many limitations. Being based on the gradient descent optimization technique, it needs a precise calculation of the gradient, which may be prohibitively expensive in many cases. Besides, it has been noticed that gradient descent-based optimization tools tend to fail when applied to large complex problems. Being trapped in local minima remains one of the main problems of any gradient descent optimization method. Other causes of failure may come from getting stuck in a plateau. It is believed that increasing the number of neurons in the hidden layer(s) of the network decreases the number of local minima, thereby increasing the chances for the algorithms to converge. A good survey of BP learning

algorithms, and gradient descent methods in general, can be found in [28], [29] and [30].

#### 7.4.2.2 Hybrid learning algorithms

In hybrid learning algorithms, two or more learning algorithms are used during the learning phase. Usually, when the term hybrid learning is used, it doesn't mean that several learning algorithms are integrated together. Simply, it means that a learning algorithm is used to learn the parameters of the antecedent and another learning algorithm is used to learn the consequent parameters of the rules. Nie and Linkens [31] integrated fuzzy inference systems with radial basis function networks. They proposed a two-phase learning algorithm (dynamical self-organizing scheme). In the first phase of the algorithm, a self-organized map (SOM) learning algorithm is used to determine the centers of the radial basis function units in the input space. Each radial basis function has identical width. The RBF units are then associated with appropriate fuzzy rules and fuzzy sets. In the second phase, a gradient descent is employed to optimally adjust the values of the output singletons. Jang [23] uses a combination of the least-squares method and the gradient descent technique for tuning his ANFIS architecture. The gradient descent is used to learn the antecedent parameters, while the least-squares method is used to learn the consequent parameters.

## 7.5 Summary

We provided in this chapter a preview of a number of recently proposed schemes integrating neural network structures with those of fuzzy logic systems. This is a newly emerging area of research where a number of findings have been quite encouraging. This is particularly true whenever the problem at hand involves the processing and the analysis of both numerical and linguistic information. Neuro-fuzzy approaches have been found to be more effective with systems involving a smaller number of fuzzy rules and a relatively small number of membership functions describing the input variables. With advances made in the field of computational software and hardware, it is anticipated that these limitations on neuro-fuzzy systems implementation will soon diminish, and systems designers should be able to tackle problems involving large amounts of linguistic and numerical knowledge. There has been in recent years a trend of combining genetic algorithm approaches for tackling optimization issues when integrating neural networks with fuzzy logic systems, and this has also been an active area of research. Some of these aspects are discussed in Chapter 8.

## Problems

1. The construction of a neuro-fuzzy system involves two essential phases: the learning structure phase (structure identification), and the learning parameter phase. What is the objective of each phase? Discuss some of the techniques used to accomplish each phase and explain their merits and limitations.

2. The partitioning technique used to identify the structure (rules) of neuro-fuzzy systems suffers from a serious limitation related to the number of inputs and the number of membership functions assigned to each input. Discuss this limitation and its impact on the model performance, and suggest possible alternative techniques to overcome this limitation.
3. The most common architecture of ANFIS is the five-layer one. However, the functionality of ANFIS can be achieved using different architectures. Discuss how the ANFIS functionality can be achieved using four-layer and one-node architectures.
4. ANFIS has the ability to model with a high degree of accuracy nonlinear mappings. Consider ANFIS to model the following two-input function:

$$f(x, y) = \frac{\cos(x)}{x} \times \frac{\cos(y)}{y}.$$

The input space of  $f(x, y)$  is to be in the range of  $[-2\pi, 2\pi] \times [-2\pi, 2\pi]$ , and three membership functions are assigned to each input. Discuss the results obtained from the model using a training set of size 200 data pairs. What is the maximum number of the rules in this case? Sketch a schematic figure for your model.

5. Repeat the previous problem, and discuss the impact of the following factors:
  - Reducing the number of rules.
  - Trying different training patterns with different sizes (e.g., 100, 125, and 150 data pairs).
  - Different step size (e.g., 0.01, 0.02, 0.05, 0.09).
6. The following three-input nonlinear function has been used by a number of researchers (e.g., Takagi *et al.*, Sugeno *et al.*, Kondo, and Jang) to verify their algorithms.

$$f(x, y, z) = (1 + x^{0.5} + y^{-1} + z^{-1.5})^2$$

If  $(x, y, z)$  are in the range  $[1, 10] \times [1, 10] \times [1, 10]$ , model this function using a feedforward backpropagation neural network and ANFIS. Compare the performance of the two models in terms of RSME, number of training epochs, and convergence time. (Choose a suitable size of training and checking data. Assume three membership functions are assigned for each input.)

7. Model the following nonlinear system using ANFIS with four-layer and five-layer structures.

$$f(x, y) = \cos(\pi x) + 2 \sin(\pi y)$$

The training and checking data can be made in the range of  $[-2, 2] \times [-2, 2]$ . Discuss and compare the results of the different models.

8. Construct an ANFIS model that is equivalent to the three-input eight-rule Tsukamoto fuzzy model. Explain how the output of each rule is induced. Sketch a schematic figure of the model and illustrate how the output of each rule is related to its membership function and the fire strength.
9. The hybrid learning procedure for ANFIS consists of a forward pass and a backward pass. Discuss this procedure and compare its performance to the backpropagation learning algorithm. Are you aware of any similar neural network that uses this type of learning?
10. Show that the radial basis function network and the ANFIS for the Sugeno model are, under some specific conditions, functionally equivalent.

1. Von Altrock, C. (1995) *Fuzzy Logic & NeuroFuzzy Applications Explained*, Prentice Hall.
2. Goonatilake, S., and Sukhdev, S. (1995) *Intelligent Hybrid Systems*, edited by S. Goonatilake and S. Khebbal, Wiley Publishing, Toronto.
3. Yamakawa, T., and Teodorescu, H. (1997) "Neuro-fuzzy systems hybrid configuration," in *Intelligent Hybrid Systems*, edited by Da Ruan, Kluwer Academic Publisher, MA.
4. Buckley, J., and Hayashi, Y. (1994) "Fuzzy neural networks, a survey," *Fuzzy sets and Systems*, vol. 71, pp. 265–76.
5. Nauck, D., Klawonn, F., and Kruse, R. (1997) *Foundations of Neuro-fuzzy Systems*, John Wiley & Sons, New York.
6. Takagi, H., and Hayashi, I. (1991) "A neural network-driven fuzzy reasoning," *International Journal of Approximate Reasoning*, vol. 5, no. 3, pp. 191–212.
7. Altrock, V. Kruse, B., and Zimmermann, H. (1992) "Advance fuzzy logic controllers techniques in automobile applications," in *Proceedings of IEEE International Conference On Fuzzy Systems*, pp. 835–42.
8. Blanco, A., Delgado, M., and Requena, I. (1995) "A learning procedure to identify weighted rules by neural networks," *Fuzzy Sets and Systems*, vol. 69, pp. 24–36.
9. Tsoukalas, L., and Uhrig, R. (1997) *Fuzzy and Neural Approaches in Engineering*, J. Wiley Interscience, USA.
10. Adeli, H., and Hung, S. (1995) *Machine Learning: Neural Networks, Genetic Algorithms, and Fuzzy Systems*, J. Wiley & Sons, New York.
11. Kosko, B. (1992) *Neural Network and Fuzzy Systems. A Dynamic System Approach to Machine Intelligence*, Prentice Hall, Englewood Cliffs.
12. Pedrycz, W., and Card, H.C. (1992) "Linguistic interpretation of self-organizing maps," *IEEE International Conference on Fuzzy Systems*, pp. 371–8, IEEE.
13. Linkens, D., and Nie, J. (1994) "Back-propagation neural-network based fuzzy controller with a self-learning teacher," *International Journal of Control*, vol. 60, no. 1, pp. 17–39. July.
14. Kohonen, T. (1989) *Self-organization and Associate Memory* (3rd edn), New York, Springer-Verlag.
15. Jang, J.R., Sun, C.T., and Mizutani, E. (1997) *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine intelligence*, Prentice Hall, Englewood Cliffs.
16. Wang, L., and Yen, J. (1999) "Extracting fuzzy rules for system modeling using a hybrid of genetic algorithms and Kalman filter," *Fuzzy Sets and Systems*, vol. 101, no. 3, pp. 353–62, Febuary.
17. Hayashi, I. Nomura, H. Yamasaki, H., and Wakami, N. (1992) "Construction of fuzzy inference rules by NDF and NDFL," *International Journal of Approximate Reasoning*, vol. 6, pp. 241–66.
18. D'Alche-Buc, F., Andres, V., and Nadal, J.P. (1994) "Rule extraction with fuzzy neural network," *International Journal of Neural Systems*, vol. 5, no. 1, pp. 1–11, March.
19. Wang, L.X. (1994) *Adaptive Fuzzy Systems and Control: Design and Stability Analysis*, Prentice Hall, Englewood Cliffs.
20. Lin, C.T., and Lee, C.S.G. (1996) *Neural Fuzzy Systems*, Prentice Hall, Englewood Cliffs.

21. Juang, F., and Lin, C.T. (1998) "An on line self-construction neural fuzzy inference network and its applications," *IEEE Transactions on Fuzzy Systems*, vol. 6, no. 1, pp. 12–32.
22. Maguire, L.P., McGinnity, T.M., and McDaid, L.J. (1997) "A fuzzy neural network for approximate fuzzy reasoning," in *Intelligent Hybrid Systems*, edited by Da Ruan, Kluwer Academic Publisher, MA.
23. Jang, J.R. (1992) "ANFIS: adaptive-network-based fuzzy inference systems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 23, no. 3, pp. 665–85.
24. Fritzke, M. (1997) "Incremental neuro-fuzzy systems," *Proceedings of Applications of Soft Computing, SPIE*, San Diego.
25. Bezdek, J.C. (1981) *Pattern Recognition with Fuzzy Objective Function Algorithms*, Plenum Press, New York.
26. Brown, M., and Harris, C. (1994) *Neuro-fuzzy Adaptive Modeling and Control*, Prentice Hall, Englewood Cliffs.
27. Rumelhart, D.E. Hinton, G.E., and Williams, R.J. (1986) "Learning internal representations by error propagation," in D.E. Rumelhart and J.L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition; vol. 1: Foundations*, The MIT Press, Cambridge, MA.
28. Baldi, P. (1995) "Gradient descent learning algorithm overview: a general dynamical systems perspective," *IEEE Transactions on Neural Networks*, vol. 6, no. 1, pp. 182–95, January.
29. Baldi, P.F., and Hornik, K. (1995) "Learning in linear neural networks: a survey," *IEEE Transactions on Neural Networks*, vol. 6, no. 4, pp. 837–58, July.
30. Perneel, C., Themlin, J.M., Renders, J.M., and Acheroy, M. (1995) "Optimization of fuzzy expert systems using genetic algorithms and neural networks," *IEEE Transactions on Fuzzy Systems*, vol. 3, no. 3, August.
31. Nie, J., and Linkens, D. (1996) "Learning control using fuzzified self-organizing radial basis function network," *IEEE Transactions on Fuzzy Systems*, vol. 1, no. 4, pp. 280–7.

# Evolutionary and soft computing



# Evolutionary computing

## Chapter outline

<b>8.1</b>	<b>Introduction</b>	<b>365</b>
<b>8.2</b>	<b>Overview of evolutionary computing</b>	<b>366</b>
<b>8.3</b>	<b>Genetic algorithms and optimization</b>	<b>372</b>
<b>8.4</b>	<b>The schema theorem: the fundamental theorem of genetic algorithms</b>	<b>375</b>
<b>8.5</b>	<b>Genetic algorithm operators</b>	<b>376</b>
<b>8.6</b>	<b>Integration of genetic algorithms with neural networks</b>	<b>388</b>
<b>8.7</b>	<b>Integration of genetic algorithms with fuzzy logic</b>	<b>390</b>
<b>8.8</b>	<b>Known issues in GAs</b>	<b>391</b>
<b>8.9</b>	<b>Population-based incremental learning</b>	<b>393</b>
<b>8.10</b>	<b>Evolutionary strategies</b>	<b>395</b>
<b>8.11</b>	<b>ES applications</b>	<b>400</b>
<b>8.12</b>	<b>Summary</b>	<b>401</b>
	<b>Problems</b>	<b>401</b>
	<b>References</b>	<b>402</b>

## 8.1 Introduction

Evolutionary computing represents another tool of soft computing techniques based on the concepts of artificial evolution. Generally speaking, evolution is the process by which life adapts to changing environments. The offspring of an organism must inherit enough of its parents' characteristics to remain viable while introducing some differences which can cope with new problems presented by its surroundings. Naturally, some succeed and others fail. Those surviving have the chance to pass characteristics on to the next generation. A creature's survival depends, to a large extent, on its fitness within its environment, which is in turn determined by its genetic makeup. Chance, however, is always a factor, and even the fittest can die in

*unlucky* circumstances. Evolution relies on having entire populations of slightly different organisms so that if some fail, others may succeed. Researchers have sought to formalize the mechanisms of evolution in order to apply it artificially to very different types of problem. The pursuit of artificial evolution using computers has led to the development of an area commonly known as *evolutionary computation* or *evolutionary algorithms*.

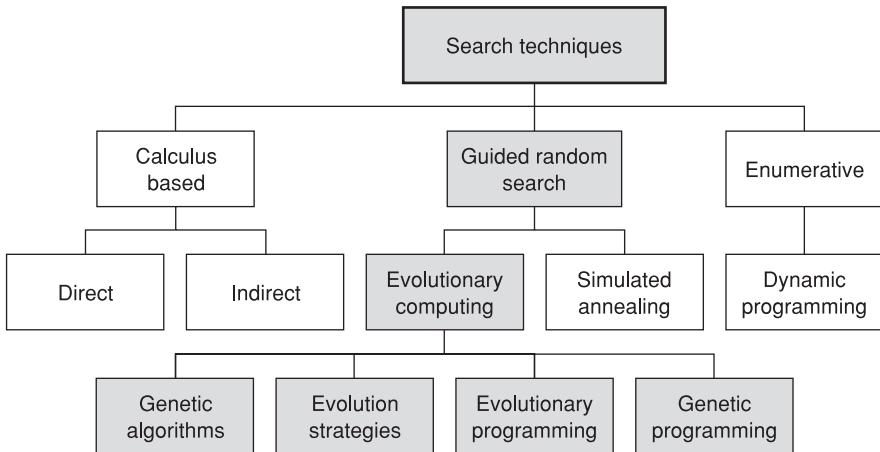
The processes involved in cellular genetics, such as a DNA replication and recombination, are very complex and are still under intensive study by researchers in the different areas of biology, chemistry, and physics. Attempting to simulate even what is well known today of these processes poses a challenging computational problem. Thus, in mapping the natural to the artificial, we must simplify and attempt to capture the essence of genetic processes in the hope that these comprise the problem-solving power we wish to use. Loosely stated, the problem addressed by natural evolution is, “what combinations of genetic traits result in an organism that can survive in its environment long enough to reproduce?”

Evolutionary computation or evolutionary computing is a broad term that covers a family of adaptive search population-based techniques that can be applied to the optimization of both discrete and continuous mappings. This computational paradigm includes such techniques as evolutionary programming, evolutionary strategies, genetic programming, and genetic algorithms. They are easily distinguished from the conventional single-point-based optimization techniques, such as gradient search methods and other directed search methods, by the fact that their search mode is performed based upon multiple positions in the search space rather than upon a single position.

## 8.2 Overview of evolutionary computing

Standard single-point-based optimization techniques operate on a point-by-point basis where the algorithm starts with a single point (current point) and a new solution is then created. The calculation of the new solution is based on a number of steps of the corresponding algorithm. Single-point-based optimization algorithms suffer from inherent demerits [1]. For instance, gradient-based methods require knowledge of the function’s derivative or partial derivatives and there is no guarantee they provide the global optimal solution. In fact, they are likely to be trapped in local minima or might even fail completely [2]. Moreover, single-point-based optimization algorithms are only efficient when the problem at hand is well defined and has a relatively simple objective function. Figure 8.1 outlines the major search techniques used today and positions evolutionary algorithms among them.

Evolutionary computing based on evolutionary algorithms, such as population-based optimization techniques, represents an excellent alternative to the single-point methods. They efficiently tackle problems that are ill-modeled or have multi-objective functions. Evolutionary algorithms are inspired by the natural selection and evolution processes. They operate on a



**Figure 8.1:** Search techniques

population of potential solutions to produce a better solution. In this process, natural principles, such as the *survival of the fittest*, are applied. The basic idea is to represent every individual of the potential solution as an array of sequences of strings, *chromosomes*. Each string in the chromosome is called a *gene* and the position of a gene is called its *locus*. The values that genes might take are called *alleles* [3].

The initial population of the potential solutions is created randomly and it evolves according to processes that are based on natural evolution, such as selection, recombination or crossover, and mutations. During these operations, which are called *evolutionary operations*, every chromosome in the population is evaluated and receives a fitness value representing an objective or a fitness function. According to their fitness values, the most successful chromosomes are selected for the crossover process to produce new offspring that might have better fitness values. The mutation process is applied to add diversity to the potential solutions. It is worth mentioning here that the population of the potential solutions is presented using some encoding mechanisms that suit the problem being tackled. The most popular encoding schemes are binary encoding, floating-point encoding and gray encoding [4]. As such, an evolutionary algorithm is characterized by the following five components:

- (1) Encoding: a mechanism to represent the population of potential solutions.
- (2) Initialization: a mechanism to create the initial population of the potential solution.
- (3) Fitness function: an objective function or evaluation function that is used to assign the fitness values to the chromosomes.
- (4) Evolutionary operators, such as crossover and mutation.
- (5) Working parameters: a set of values of the different parameters such as population size and chromosome length.

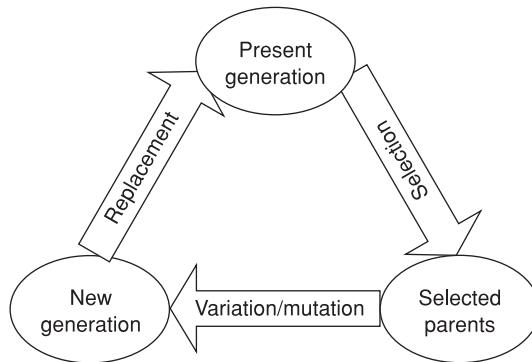


Figure 8.2: Basic evolution cycle

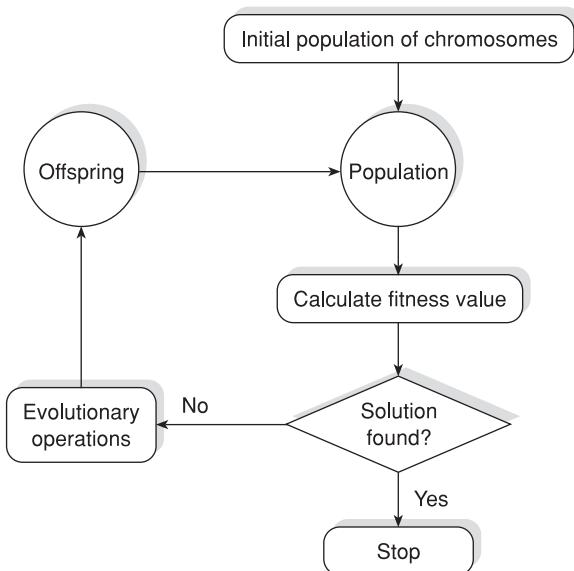
As is the case for all evolutionary processes found in nature, the implementation of evolutionary algorithms comprises several iterations of the basic *evolution cycle*. The concept of evolution cycle is shown in Figure 8.2. With the appropriate choice of a set of algorithmic strategies and related parameters, we try to improve the quality of solutions offered by the system after a reasonably small number of iterations and samples have been chosen from a large search space.

Different evolutionary computing techniques incorporate variation of the basic evolution cycle in terms of presentation models and specific combinations of evolutionary operations (such as selection, crossover, and mutation). The interesting aspect pertaining to the implementation stage is to balance out two opposite operations. On the one hand, the selection operation intends to reduce diversity of the sample population (a small set of possible solutions from a vast number of solutions). On the other hand, the crossover and the mutation operators try to increase the diversity of the population. These two opposite evolutionary operations enable the algorithm to improve the quality of the sample solutions while exploring the whole search space.

Being a special family of optimization algorithms, evolutionary algorithms should be tackled and used as an alternate tool only if their usage makes a difference in terms of:

- High rate of convergence
- Good quality of evolved solution
- Reasonable computational requirements.

So far, no general analysis framework has been proposed to analyze the general form of evolutionary algorithms. However, some specific variations or implementations could be tackled along two lines of investigation: theoretical and empirical approaches. The theoretical approach attempts to discover mathematical truths of algorithms that will hold within a reasonably broad domain of applications. The empirical approach, on the other hand, attempts to assess the performance of an implementation in a specific domain



**Figure 8.3:** Illustration of the basic structure of an evolutionary algorithm

of application. Both methods have advantages and disadvantages, and in practice they should be used as a complementary means for designing and tuning the specific instance of an algorithm. The flowchart in Figure 8.3 illustrates the basic algorithmic structure of a general evolutionary process.

As mentioned earlier, evolutionary algorithms depict a wide representation of techniques that include evolutionary programming, evolutionary strategies, genetic programming, and genetic algorithms. The following highlights briefly the concepts and the features of each one of these techniques. More detailed analysis is dedicated later in the chapter to genetic algorithms and evolutionary strategies, given their high rate of usage among scientists and engineers. It should be noted here that there are many hybrid systems that incorporate various features of the different evolutionary algorithms. Further details on such systems can be found in [5] and a good survey of evolutionary computing can also be found in [6,7,8,9].

### 8.2.1 Evolutionary programming

Evolutionary programming (EP) techniques, developed by Fogel [10] in the late 1980s, are stochastic-based optimization methods that aim at generating machine intelligence that has the ability of predicting changes in the environment. The basic EP method involves three steps that are repeated until a termination condition is attained. The three steps are summarized as follows:

- An initial population of potential solutions is created randomly.
- Each solution is replicated into a new population and the new chromosomes are mutated according to the mutation distribution.

- The new chromosomes are evaluated based on their fitness-function values. A selection process is then applied to retain the survival solutions of the new generation.

In evolutionary programming the chromosomes are represented as finite state machines (FSMs). The objective is to optimize these FSMs in order to provide a meaningful representation of behavior based on the interpolation of the symbol [5]. Typically, EP does not use crossover operators. The main process here is the mutation operation, which is used to randomly create new offspring from the parent population [3]. There exist five possible mutation operators:

- Modify an output symbol.
- Modify a state transition.
- Add a state.
- Delete a state.
- Change the initial state.

One of these mutation operators is selected based on a mutation distribution (usually a probability distribution). Furthermore, more than one mutation operator can be applied to one parent.

### 8.2.2 Evolutionary strategies

Evolutionary strategies (ES) were originally developed by Rechenberg [11] as a method to solve continuous optimization problems. They make use of a population of a size equal to one with one single genetic operation, i.e., mutation. Schwefel [12] utilized the real-valued form to represent the chromosomes and compared the performance of the ES with more traditional optimization techniques. Moreover, he extended the population size to more than one and introduced other operators such as selection and recombination. For a survey of evolutionary strategies and their applications, one may consult [13].

### 8.2.3 Genetic programming

In genetic programming (GP), the potential solutions of the population are the parse trees of computer programs where the fitness values are measured by running them [14]. A GP may be considered as a subset of genetic algorithms with the difference in terms of the solutions representation. A GP is usually implemented using the following four steps:

- Initialize the population of potential solutions (parse trees of computer programs) of functions and terminals. These functions and terminals are the alphabet of the computer programs.
- Execute each program and assign a fitness value to it according to how well the program does in solving the problem.

- Generate new offspring of the computer programs as follows:
  - (a) Copy the computer program that has the best fitness value.
  - (b) Create a new computer program using the mutation operator.
  - (c) Apply a crossover operator to create new computer programs.
- The program with the highest fitness value is considered to be the result of the genetic programming.

In the steps above, it should be noticed that the mutation operator is applied before the crossover operator. Mutation in genetic programming has two types. In the first, a function can only replace a function and a terminal can only replace a terminal. In the second type, an entire sub tree of the parse tree population can replace another sub tree. Here, the mutation point is randomly chosen. In the crossover operation, two parental programs are probabilistically chosen from the population based on their fitness values. After the point of crossover has been randomly determined for both chromosomes, a sub tree from the first one is substituted with a sub tree from the second. In this case, the first chromosome is called the *receiving* chromosome and the second one is called the *contributing* chromosome [15]. In GP, the crossover operation is an essential process to introduce diversity to the population.

#### 8.2.4 Genetic algorithms

Genetic algorithms (GAs) represent another important class of evolutionary computing techniques. GAs are non-comprehensive search techniques used to determine among other things the global optima of a given function (or a process) that may or may not be subject to constraints. The origin of GAs dates back to the early 1950s when a group of computer scientists and biologists teamed up to simulate the behavior of a class of biological processes. But it was only later in the early 1970s that Holland and his associates introduced the methodology in a more formal and tractable way. Holland was able to show using the schema theory [16, 17] that genetic algorithms have sound theoretical roots and they are able to accurately solve a wide range of optimization problems. This is done through a procedure inspired from the biological process of evolution and the survival of the fittest concept. GAs have seen in recent years wide interest from researchers in the field of engineering mathematics, connectionist modeling and approximate reasoning, to name a few. The search procedure of GAs is stochastic in nature and doesn't usually provide the exact location of the optima as some other gradient-based optimization techniques do. However, GA-based techniques possess two attractive features putting them at an advantage with respect to their derivative-based counterparts. In fact, given their discrete search nature, they could be easily applied to continuous as well as to discontinuous functions. Moreover, while GAs may not provide for the mathematically exact solution of a given optimization problem, they usually outperform gradient-based techniques in getting close to the global optima and hence avoid being trapped in local ones.

Given their growing use in an ever-increasing broad set of applications, genetic algorithms and evolutionary strategies have become major tools of evolutionary computing. The remaining part of this chapter is dedicated to the major characteristics and features of these two powerful soft computing tools. For details on other tools of evolutionary computing, the reader may wish to consult the work in [5] and [14].

### 8.3 Genetic algorithms and optimization

A large number of techniques have been proposed in the literature to solve optimization problems for smooth functions and piecewise smooth functions. Many of these techniques use the gradient of the function to find the best direction in searching for the optima. Well known among them are the *path-following-based* methods and the *integral-based* procedures. But since the gradient of a function is always zero whether it falls at a local or at a global optimum, the algorithm always terminates without giving an indication of whether or not it has reached the global optima. As such, all gradient-based optimization methods have no guarantee of providing the global optima. Several methods have been suggested to overcome this problem, including the path-following methods and the integration-based methods. But while alleviating some of the issues, most of these techniques still rely on the gradient of the function.

Techniques termed *genetic algorithms* were formulated first by Holland [16]. Based on biologic evolution and on the survival of the fittest principle, these techniques have been proposed as effective tools for dealing with global optimization problems. They are derivative-free based techniques, and as such could be easily applied to smooth functions, simply continuous functions, or even discontinuous functions. They are based on the evaluation of the function at a set of points in the function's variable space, usually chosen randomly within the search range. This feature makes them less vulnerable to local optima, and makes them good candidates for solving global optimization problems. The solution is obtained on the basis of an iterative search procedure, which mimics to a certain extent the evolution process of biological entities. The end result of this process, which starts with a randomly selected population of individuals, is a population with a strong survivability index, something better known as the *survival of the fittest* principle. This translates into finding the location of the point(s) at which the function is maximal. In a more formal way, let us suppose the goal is to maximize a function  $f$  of  $m$  variables given by:

$$f(x_1, x_2, \dots, x_m): \Re^m \rightarrow \Re \quad (8.1)$$

At the end of the optimization process one gets:

$$f(x_1^*, x_2^*, \dots, x_m^*) \geq f(x_1, x_2, \dots, x_m) \quad (8.2)$$

where  $(x_1^*, x_2^*, \dots, x_m^*)$  represents the vector solution belonging to the search space(s) taken here as  $\Re^m = (\Re \times \Re \dots \times \Re)$ . It is presumed that the function  $f$  takes positive values. If this is not the case, then one may wish to add a bias term into the function to make it positive over all the search space. In the case of minimization, one might pose the problem of optimization as that of maximization of a function  $h$ , which is the negation of  $f$  all over the search space. In other words, maximize the function  $h$  such as:

$$\min f(x_1, x_2, \dots, x_m) = \max h(x_1, x_2, \dots, x_m) = \max(-f(x_1, x_2, \dots, x_m)) \quad (8.3)$$

The basic idea of GAs is to choose first a random population in the range of optimization, with a fixed size  $n$  ( $n$  usually depends on the search range, the accuracy required and the nature of the function itself). Using the so-called *binary encoding* procedure, each variable is represented as a string of  $q$  binary digits. This leads to a population of elements represented by a matrix of  $n$  rows and  $qm$  rows. A set of “genetic” operators is then applied to this matrix to create a new population at which the function  $f$  attains increasingly larger values. The most common operators that have been used to achieve this task are: *Selection*, *Crossover* and *Mutation*. These operators are described later in detail following the definition of two terms used most often in the jargon of genetic algorithms: the genotype of a population and its fitness value.

### 8.3.1 Genotype

When attempting to map *natural evolution* into the framework of *artificial evolution*, we must first consider the “data” for the system. In nature, these data consist of living creatures. Each individual represents a potential solution to the problem of survival. Similarly, in genetic algorithms, we consider a set of potential solutions, which are referred to collectively as a *population*. Each single solution is called an *individual*. Each individual in nature has a form determined by its DNA. Its collection of genetic traits is commonly known as a *genotype*. In genetic algorithms, the term genotype is used to describe the encoding of a problem solution represented by an individual. Thus, each individual has a genotype, which encodes a solution. Many individuals in a population may have the same or similar genotypes. In the GA literature, an individual’s genotype is often referred to as its *chromosome*. Genotypes in genetic algorithms are typically represented by strings, sometimes called bits or characters. Each element of the string represents a *gene*, which is a single unit of genetic information. In nature genes control, directly or indirectly, various traits in the individual. For example, in humans there are genes to determine eye and hair colour, and genes for determining other characteristics. It is important to note that in nature, several genes often collectively determine a physical trait, and that they are not necessarily independent. This is true in genetic algorithms as well, where a solution encoding may make use of several interacting genes. Each gene has one or more

possible values known as *alleles*. One can imagine that humans have a hair colour gene with an allele for each colour: brown, blond, black, red, etc. The reality in humans is far more complicated than this but it serves as an illustration of the idea. The number of alleles for a specific gene is essentially fixed in nature, and in artificial evolution it is determined by the encoding of solutions. The simplest genes are binary, having only two alleles. This means they can be represented by a single bit. However, some genes may have several alleles and are represented using characters. In genetic algorithms, the number of genes in the genotypes for a particular problem is usually fixed. There have been some applications which employ variable-length genotypes but these will not be addressed here. Since the genotype is intended to express a solution to a specific problem, the genes and alleles must be designed for that problem and must express the various components of a potential solution. The design of the genotype structure for a particular application is one of the two most difficult and important parts of using genetic algorithms. Unfortunately, there are no straightforward, general-purpose approaches. The task is highly dependent on the problem and, in many cases, on the particular kind of genetic algorithm one wishes to employ.

### 8.3.2 Fitness function

To use genetic algorithms, it is necessary to provide a means for evaluating the *value* or *goodness* of a particular solution. In nature, this is frequently thought to be the *fitness* of a creature (as in the popular concept of *survival of the fittest*), referring to its relative ability to survive in its environment. A *fit* creature must be able to find food and shelter, must be able to endure the local temperature and weather, and detect and evade predators. If one creature is able to do this better than another, we say that it is “fitter.” This leads us to the concept of a *fitness function*, which measures the *fitness* of a particular solution. In genetic algorithms, fitness functions are also called *objective functions*. An objective function takes a genotype as its parameter and typically gives a real-valued result that represents the *fitness* or *goodness* of the solution. Generally speaking, this fitness is comparative rather than absolute, and serves to differentiate among many solutions rather than to discover the ideal solution. The ability to compare solutions is, in most cases, essential to the operation of a genetic algorithm.

In a number of cases, the choice of objective function is quite obvious. In the equation minimization problem described earlier, the genotype is a set of parameters for the function, and the objective function is simply the value of the equation being minimized, given the genotype. In this case, a lower result from the objective function represents a better solution to the problem. In many cases, however, a good objective function is more difficult to construct and heuristic approaches must be used. If one considers the manufacturing process, for instance, a genotype could consist of a set of values for various control parameters of the process, but the objective function is more difficult to quantify. One possibility is to measure the yield from the process (the more

yield the better). Another might be to measure the quality of the product. Still another might be the cost of the inputs to the process. These quantities might be difficult or slow to measure and we would probably wish to judge a solution by several of these criteria rather than just one. The objective function is not only problem-specific, but is also inherently specific to the genotype used to represent the solutions. Changing the structure of the genotype generally requires changing the evaluation of the objective function, and vice versa. For many problems, the objective function may be expensive to compute, or may simply offer an unreliable comparison of two solutions. In some cases, the value of an optimal solution may not be known, even though the objective function itself is known. The objective function may not be a mathematical function at all, but a measurement taken of the solution's performance in a simulation or real-world experiment. This puts the design of the objective function among the more difficult and important steps in working with genetic algorithms.

## 8.4 The schema theorem: the fundamental theorem of genetic algorithms

The schema (plural schemata) theorem, introduced by Holland [26] in the mid-1970s, represents the formal foundation for genetic algorithms. The basic idea behind the schema theorem is to represent a template or prototype of symbols: 0, 1, and \* (known also as a *wildcard* or *don't care* symbol) to generate populations of chromosomes. This idea is then generalized as follows: if the chromosomes of a population are encoded using symbols belonging to the alphabet  $\Omega$  (an alphabet is a set of symbols), then the schemata can be expressed as all the chromosomes whose symbols belong to  $\Omega \cup \{*\}$ . The size of the generated population depends directly on the number of *don't care* symbols in the schema. A schema containing  $n$  *don't care* symbols matches  $2^n$  chromosomes, and for any given schema, the generated chromosomes will share with it some of its traits or characteristics.

For instance, the schema (1\*0\*1) matches the following four ( $2^2$ ) chromosomes:

10001 10011 11001 11011.

In general, schemata are described by the two following important characteristics:

- (1) The schema *order*,  $o(s)$ , describes the number of non-*don't care* symbols in the schema. That is, the number of 0s and 1s in the case of binary encoding. The schema order is an important characteristic since it describes the generality of the schema. The smaller *order* a schema has, the more general it is; i.e., it can generate a larger population size.

- (2) The schema *defining length*,  $\alpha(s)$ , is the distance between the furthest two non-*don't care* symbols. This characteristic describes the compactness of the schema or the likelihood that two symbols in a chromosome will stay together following the crossover process. For example, the closer two symbols are, the more likely they will stay together after the crossover operation.

The next example illustrates the concept of the schema order and the notion of length for different schemata.

### Example 8.1

Schema: (s)	Order: $o(s)$	Defining length: $\alpha(s)$
1*****	1	0
**101***	3	2
*1*01***	3	3

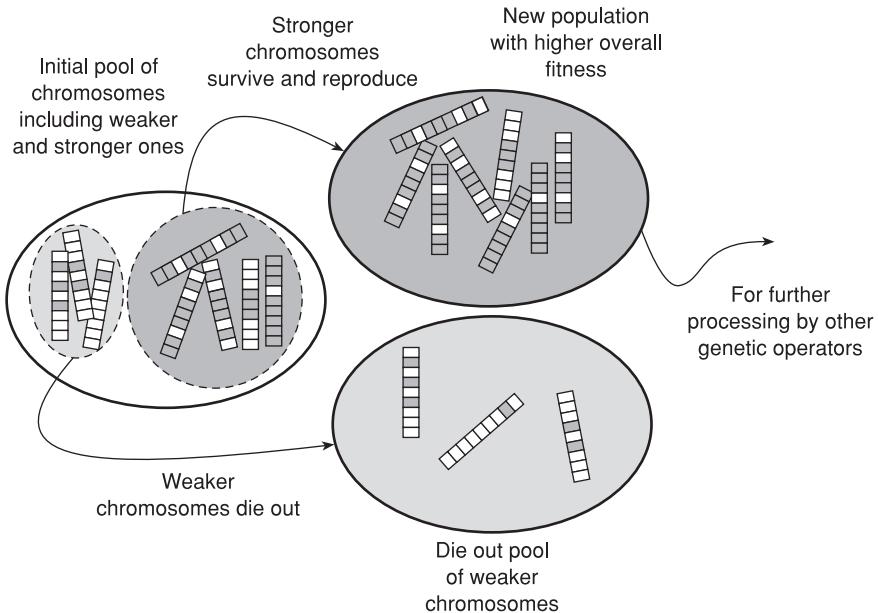
The basic schema theorem can then be interpreted using the following equation, which relates the schema fitness of the current generation to the expected chromosomes in the new generation:

$$M(H, t + 1) \geq M(H, t) \frac{f(H)}{F_{tot}} (1 - p_{xo} - p_m) \quad (8.4)$$

where  $M(H, t)$  is the number of chromosomes in population  $t$  that matches the schema  $H$  and  $f(H)$  is the average fitness of the chromosomes with schema  $H$ .  $F_{tot}$  is the average fitness of the whole population,  $p_{xo}$  and  $p_m$  are the probabilities of crossover and mutation, respectively. The fitness of a schema is the average fitness of chromosomes generated from that schema. It should be noticed here that schemata fitness is more accurate if the order of the schemata is low. This is due to the fact that low order schemata generate more chromosomes than their higher order counterparts.

## 8.5 Genetic algorithm operators

The most important aspect of an evolutionary process pertains to the way a composition of a population changes. In nature, we commonly look at three major forces: *natural selection*, *mating*, and *mutation*. The equivalents to these forces in artificial evolution are: *selection*, *crossover*, and *mutation*. Collectively, they form a large group of processes, which act on individuals, sets of individuals, populations, and genes, and are known as *genetic operators*.



**Figure 8.4:** Illustration of the selection operation in GA (the darker the chromosome, the higher its fitness value)

### 8.5.1 Selection

This procedure is applied to select the individuals that participate in the reproduction process to give birth to the next generation. Selection operators usually work on a population, and may serve to remove weaklings, or to select strong individuals for reproduction. In general, selection operators are stochastic, probabilistically selecting good solutions and removing bad ones based on the evaluation given to them by the objective function. There are several heuristics in this process, including the elitist model, where the top 10 to 20 individuals of the population are chosen for further processing; the ranking model, where each member of the population is ranked based on its fitness value; and the roulette wheel procedure where each individual  $i$  is assigned a probability  $p_i$  to be chosen for reproduction, after which the cumulative probability  $c_i = \sum_{j=1}^i p_j$  is calculated for each  $i$ . In the last model, an individual is selected if  $c_i$  becomes greater than a random number  $r$  selected *a priori*. Figure 8.4 illustrates the process of selection in GAs for an initial population of eight chromosomes.

### 8.5.2 Crossover

Crossover is derived from the natural phenomenon of mating, but refers most specifically to *genetic recombination*, in which the genes of two parents are combined in some more or less random fashion to form the genotype of

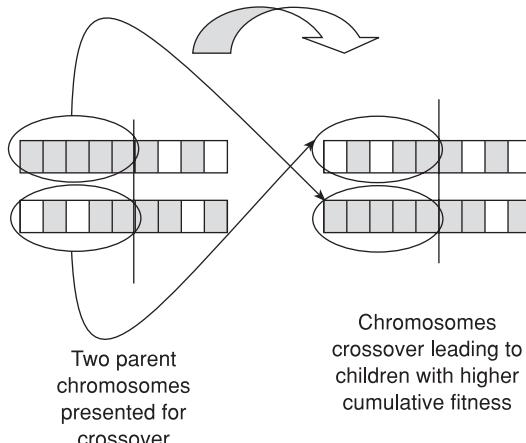


Figure 8.5: Crossover operation for two chromosomes

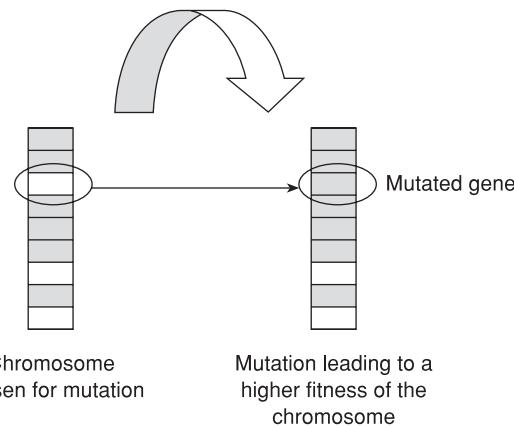
a child. Note again that this process involves randomness, and thus most *crossover operators* randomly select a set of genes from each parent to form a child's genotype. There exists in the literature a number of crossover operators, but the most common one consists of choosing a number of points (one for simple crossing) in the binary strings of the two parents to create the offspring by exchanging the parents' gene sequences around these points. This is illustrated in Figure 8.5.

### 8.5.3 Mutation

While the crossover operation generates new combinations of genes, and therefore new combinations of traits, mutation can introduce completely new alleles into a population. It has been widely recognized that mutation is the operator that creates completely new solutions while crossover and selection serve to explore variants of existing solutions while eliminating bad ones. In nature, mutations are random alterations in genetic material resulting from chemical or radioactive influences, or from *mistakes* made during replication or recombination. In genetic algorithms, we typically design *mutation operators* that select genes in an individual at random and change the allele. In some cases, the change in an allele is random too, simply selecting a different allele from those available for that gene. In other cases, especially if the problem structure suggests such a mechanism, alleles are mutated in a deterministic fashion, and only the selection of genes is made random. In terms of implementation, mutation consists of randomly changing one or more parts of a string. This is done by flipping the digits starting from a randomly chosen order as shown in Figure 8.6.

### 8.5.4 Mode of operation of GAs

For each population, the GA operators are applied to the chromosomes to lead to the new generation of individuals, ameliorating in the process the best



**Figure 8.6:** Mutation operation over a single gene

fitness among the individuals of the generation. The process is terminated after a fixed number of generations has been reached, or when the best fitness value is no longer ameliorated from one generation to the next. Other less standard techniques have been proposed for constructing the new generations; in particular, *duplication* and *inversion*, which were proposed by Wienholt [18].

Unlike other search-based optimization procedures such as Hill Climbing or Random Search, GAs have consistently achieved good performance in terms of balancing between the two conflicting objectives of any search procedure, which are the *exploitation* of the best solution and the *exploration* of the search space. GAs have also a number of other interesting features that differentiate them from other search techniques:

- GAs work with a coded version of the parameters.
- They start the search procedure from a population of points (solutions) not from a single point, which makes them suitable candidates for parallel processing.
- They use stochastic reproduction schemes instead of deterministic ones.

The search mechanism of GAs follows a step-by-step algorithm characterized by an iterative procedure having as its main goal the improvement of the fitness measure of the individuals. For a simple optimization problem and at a given iteration, these steps are initialization of the population, the encoding of the variables, evaluation of the fitness measure at each generation and application of the genetic operators that lead to the new generation.

The encoding mechanism permits the representation of the variables belonging to the search space in such a way that it carries all the required information (chromosomes) characterizing the fitness measure of a particular population. The two most used formats for encoding the variables are the binary format and the floating-point format. Traditionally, floating-point format has been used to represent the individuals of a population when dealing with

optimization problems involving multidimensional and high-precision numerical solutions. Binary format representation on the other hand is simply not appropriate for this type of problem given the huge search space that may result. For problems that do not require a high level of precision and where the range space is small, binary format representation has consistently provided acceptable solutions. Binary representation of individuals requires encoding (into binary) and decoding (into decimal). This is a clear disadvantage in terms of computational time required. However, a binary encoded chromosome has a lot more elements (every bit is considered to be an element) than an individual represented by a floating-point has. Indeed in genetic algorithms, individuals with more elements carry more information than their counterparts with fewer elements. In using the floating-point representation, each chromosome is created by putting all the floating-point numbers one after another. This means the encoding and decoding steps don't have to take place. For the sake of illustration, only binary representation is considered in the following description. For further details on different format representations, one may consult [19].

For binary encoding, a binary vector composed of  $n$  bits is used as a chromosome to represent a real-valued parameter (variable)  $x$ .<sup>\*</sup> For a multi-variable function with  $m$  parameters ( $x_i$ ,  $i = 1, \dots, m$ ), the chromosome becomes of length  $\sum_{i=1}^m n_i$  where  $n_i$  represents the precision of the variable  $x_i$ , which can take different values for each of the variables depending on the problem's specifications. Suppose that the objective function  $f$  has  $m$  parameters (function with  $m$  variables):

$$f(x_1, x_2, \dots, x_m): D_1 \times D_2 \times \dots \times D_m \rightarrow \mathfrak{R} \quad (8.5)$$

If each of the variables  $x_i$  has its domain of variation specified within the interval  $D_i = [\alpha_i, \beta_i]$ , and if the required precision is given by  $\gamma$  decimals (chosen here to be the same for each variable  $x_i$ ), then the domain  $D_i$  should be divided into  $(\beta_i - \alpha_i)10^\gamma$  equal-size intervals. If  $n_i$  is taken as the smallest integer such that  $(\beta_i - \alpha_i)10^\gamma + 1 \leq 2^{n_i}$  then each parameter  $x_i$  could be encoded as a binary string of length  $n_i$ . The  $\sum_{i=1}^m n_i$ -long total chromosome  $X$  is hence created by concatenating all encoded parameters one after another:  $X = [x_1, x_2, \dots, x_m] \triangleq [b_{11}, b_{12}, \dots, b_{1n_1}, \dots, b_{m1}, b_{m2}, \dots, b_{mn_m}]$ , where  $b_{ij}$  represents the  $j$ -th bit of the  $i$ -th variable  $x_i$ . By decoding the chromosome again, the original values of the parameters are then obtained. The following formula can be used to decode the value of each parameter  $x_i$ :

$$x_i = \alpha_i + \text{decimal}(\text{string}_i) \cdot \frac{\beta_i - \alpha_i}{2^{n_i} - 1} \text{ and } \text{string}_i = b_{i1} b_{i2} \dots b_{in_i} \quad (8.6)$$

where the argument of *decimal* is the binary representation of the parameter  $x_i$  and *decimal(string<sub>i</sub>)* represents the decimal representation of the binary string.

---

\* Parameter and variable in the context of this section are equivalent.

### 8.5.5 Steps for implementing GAs

We outline next the procedure for carrying out a genetic algorithm [26] for the optimization of a given function  $f(x)$  taken here as  $fit(x)$ .

*Step 1* Encode the variables of the algorithm as binary chromosomes  $v_i$  where ( $i = 1, 2, \dots, p$ ) and  $p$  is the population size of the possible solutions.

*Step 2* Initialize the population of chromosomes.

*Step 3* Perform the following steps until the predefined condition is achieved:

- Evaluate (calculate) the fitness function values  $fit(v_i)$  for each chromosome  $v_i$  ( $i = 1, 2, \dots, p$ ).
- Select a new generation (using the roulette wheel for example).
- Calculate the total fitness of the population:

$$F = \sum_{i=1}^p fit(v_i) \quad (8.7)$$

- Calculate the probability of selection  $p_i$  for each chromosome  $v_i$  ( $i = 1, 2, \dots, p$ ):

$$p_i = \frac{fit(v_i)}{F} \quad (8.8)$$

- Calculate a cumulative probability  $q_i$  for each chromosome  $v_i$  ( $i = 1, 2, \dots, p$ ):

$$q_i = \sum_{j=1}^i p_j \quad (8.9)$$

- Generate a random float number  $r \in [0, 1]$ .
- If  $r < q_1$  then select  $v_1$ ; otherwise select  $v_i$  ( $2 \leq i \leq p$ ) such that  $q_{i-1} < r \leq q_i$ .
- Apply genetic operators (crossover, mutation).
  - Select chromosomes for crossover according to the crossover probability,  $p_c$ .
  - Choose a chromosome for mutation according to mutation probability,  $p_m$ .
- Choose the new offspring as the current population.

*Step 4* Go back to step 3 if the optimization requirement is not attained.

These different steps are summarized in the schematic block diagram of Figure 8.7.

### 8.5.6 Search process in GAs

The classic genetic algorithm, as stated above, starts with a completely random population of solutions. Iteratively, it selects mating couples with a

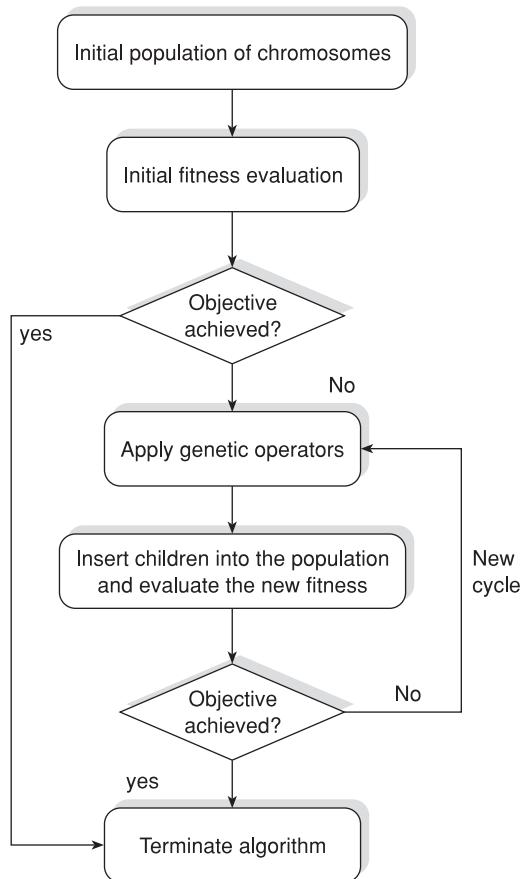


Figure 8.7: Schematic representation of a genetic algorithm

bias towards the stronger individuals in the population. These couples “mate” producing offspring that are either identical to the parents or that inherit part of their traits from each parent. These children are then subjected to mutation, which may or may not randomly alter their genetic makeup. The entire process is then repeated on the children, forming a series of generations.

As the algorithm proceeds, the weaker solutions tend to be discarded and hence are not selected for breeding and are unable to pass on their genes. Stronger solutions may breed many times, producing many children that share their traits. The probabilistic crossover means that some of the children will be exact copies, thus increasing the presence of strong solutions in the population. Many will be the result of crossover, inheriting traits from both parents. This recombination may produce a better individual than either parent, but can also produce worse solutions. The hope is that the best traits of two parents may be combined to create a better individual. However,

the probability of this happening is dependent on both the problem and its representation (genotype structure).

In the GA process, mutation serves to introduce alleles and combinations of alleles that may have been selected out or never existed in the population. Since selection and crossover can only recombine existing material, mutation serves an essential role by introducing and reintroducing genetic material which leads to even better solutions. Mutation often results in a worse solution, as can be observed in nature, but its absence generally leads to stagnation. This leads to a key feature of genetic algorithms, which is related to balancing between *exploration* (the examination of new areas of the problem space) and *exploitation* (the examination of well-known areas of the problem space). In a number of cases, there are several regions containing good solutions. A purely exploratory algorithm (e.g., random search) might search widely through the problem space, but, having found a decent solution, might never search “near” that solution to see if better solutions exist close by. By contrast, a purely exploitative algorithm (e.g., a hill-climber), having found a decent solution, might search only in that area, and may never discover a much better solution far away in the problem space. GAs feature exploration (through mutation and, to some extent, crossover) and exploitation (through selection and again through crossover). The degree to which the GA explores or exploits is determined by its parameters and its structure. These are related to the population size, the crossover rate, and the mutation rate. They also depend on the termination conditions, the genotype structure and the choice of fitness function. This makes GAs very sensitive to the choice of their parameters and hence a lot of planning and experimentation is required to produce the desired results. Some of the issues encountered in GA problem-solving are outlined in section 8.8.

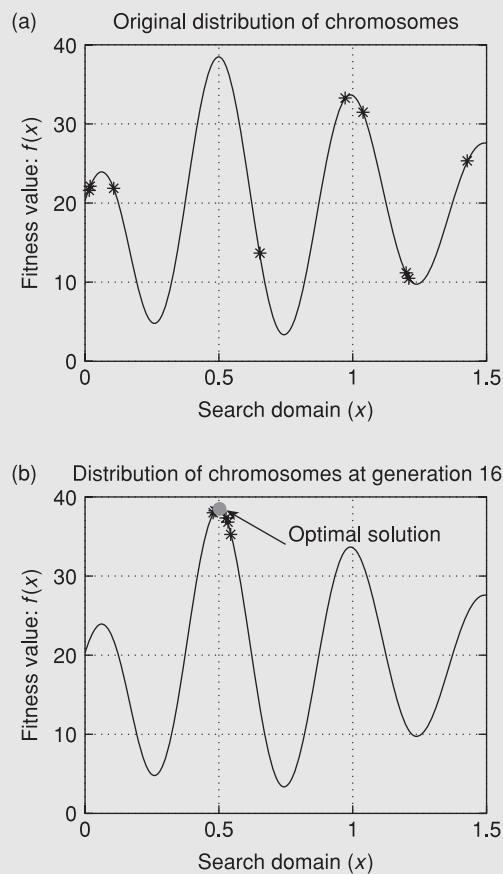
### Example 8.2

It is desired to find the maximum of a function  $f$  of one variable  $x$  given by

$$f(x) = 20 + 100 * \cos(4\pi x) \exp(-2x) \quad (8.10)$$

The range of optimization is chosen to be  $[0, 1.5]$  and the required precision is given by three decimals. Using the formulae developed in section 8.5.4 relating accuracy, number of bits in a binary representation and the boundary of the search interval, we find that each chromosome is represented by  $n_i = 11$  binary digits with  $n_i$  satisfying the smallest integer such that:  $((1.5) - (0)) \cdot 10^3 \leq 2^{n_i} - 1$ . Moreover a simple plot of the function shows that it reaches its maximum at the value corresponding to  $x = 0.5$  as shown in Figure 8.8(a).

Using the steps outlined in the previous section with a population of first parent chromosomes equal to 10, a crossover probability of 0.25 and a mutation probability of 0.1, we find the following results summarized in the following tables.



**Figure 8.8:** (a) Original distribution of chromosomes on  $f(x)$ . (b) Chromosome distribution after 16 generations

*Initial population:*

**Table 8.1:** Initial population and chromosome encoding

Chromosome $x_i (i = 1, \dots, 10)$	Binary encoding for $x_i$	Decimal value of $x_i$	Fitness	Probability for selection $p_i$
$X_1$	00010010100	0.108	21.802	0.1803
$X_2$	11001100111	1.201	11.121	0.0552
$X_3$	11001101001	1.202	11.023	0.0547
$X_4$	10100110001	0.973	33.144	0.1646
$X_5$	11001110111	1.212	10.428	0.0518
$X_6$	01101111101	0.654	13.622	0.0677
$X_7$	00000010110	0.016	21.529	0.1069
$X_8$	11110100000	1.430	25.248	0.1254
$X_9$	10110001011	1.039	31.402	0.1560
$X_{10}$	00000011110	0.022	22.024	0.1094

The total fitness of the whole population is given as 201.347. The relatively large probability for selection  $p_i$  for chromosomes shows that  $X_4$ ,  $X_9$ , and  $X_{10}$  will produce similar chromosomes.

*The selection operation leads to the following population:*

**Table 8.2:** Improved population after selection is made

Chromosome $x_i (i = 1, \dots, 10)$	Binary encoding for $x_i$	Decimal value	Fitness
$X_1$	00000011110	0.022	22.024
$X_2$	00000011110	0.022	22.024
$X_3$	11001101001	1.202	11.023
$X_4$	10100110001	0.973	33.144
$X_5$	10110001011	1.039	31.402
$X_6$	10110001011	1.039	31.402
$X_7$	10100110001	0.973	33.144
$X_8$	10110001011	1.039	31.402
$X_9$	10100110001	0.973	33.144
$X_{10}$	00000010110	0.016	21.529

The new total fitness becomes equal to 270.241, which is an already substantially increased fitness over that of the original population.

During crossover, each selected chromosome is assigned a cumulative probability  $q_i = \sum_{j=1}^i p_j$  as described in the previous steps of the GA procedure. For this particular generation, this leads to a crossover between chromosome 1 and chromosome 5. As such the child chromosome 1 becomes expressed in bit strings as 00000001011 and child chromosome 5 becomes expressed as 10110011110. The mutation operation on the other hand has affected chromosome 10 only, which has now become expressed as 10110011110. This is expressed in Table 8.3.

*The new population after crossover and mutation:*

**Table 8.3:** Population of chromosomes following crossover and mutation

Chromosome $x_i (i = 1, \dots, 10)$	Binary encoding for $x_i$	Decimal value	Fitness
$X_1$	000000010111 CO*	0.008	20.789
$X_2$	00000011110	0.022	22.024
$X_3$	11001101001	1.202	11.023
$X_4$	10100110001	0.973	33.144
$X_5$	10110011110 CO*	1.053	29.996
$X_6$	10110001011	1.039	31.402
$X_7$	10100110001	0.973	33.144
$X_8$	10110001011	1.039	31.402
$X_9$	10100110001	0.973	33.144
$X_{10}$	10110011110 M*	0.043	23.251

\* CO and M denote that crossover and mutation operations have occurred at that particular chromosome

The new overall fitness has now become equal to 269.323, which is not really an improvement over the previous operation (population after selection). During the next generations, however, the crossover and mutation would tend to improve even further the original fitness of the chromosomes. In this particular example and after only 16 further generations, the maximum of the function is reached with a high degree of accuracy. This is summarized in Table 8.4.

*The population of chromosomes after 16 generations:*

**Table 8.4:** Population after 16 generations

Chromosome $x_i (i = 1, \dots, 10)$	Binary encoding	Decimal value	Fitness
$x_1$	01010010001	0.481	37.883
$x_2$	01011101011	0.547	35.163
$x_3$	01010010001	0.481	37.883
$x_4$	01010010001	0.481	37.883
$x_5$	01011011001	0.534	36.684
$x_6$	01010011001	0.487	38.154
$x_7$	<b>01010101011</b>	<b>0.505</b>	<b>38.393</b>
$x_8$	01011010001	0.528	37.213
$x_9$	01000011001	0.383	24.127
$x_{10}$	01010011001	0.487	38.154

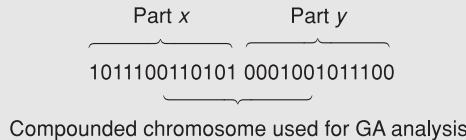
The new cumulative fitness of the population after 16 generations becomes equal to 361.540, which shows that most chromosomes have now gathered around the optimum value of the function as shown in Figure 8.8b. It is approximated in this case to a very high degree of accuracy by chromosome 7, whose decimal value is given by 0.505 and leads to a maximum value of  $f(x) = 38.393$ . These values are very close to the solution of the problem given by the pair (0.5, 38.393). During the simulation of this particular problem, further generations didn't necessarily provide better solutions than the one reached in generation 16. As such it is customary to store past runs of the algorithm. This is particularly useful in case the stopping criterion is determined by the number of generations taking place instead of the attained accuracy.

### Example 8.3

We need to optimize here the well-known *peaks* function  $g(x, y)$  given as:

$$g(x, y) = 3(1 - x)^2 e^{(-x^2 - (y+1)^2)} - 10((x/5) - x^3 - y^5) e^{(-x^2 - y^2)} - (1/3) e^{(-(x+1)^2 - y^2)} \quad (8.11)$$

The range of optimization representing the 2-D search space is taken here as  $(x, y) \in [-3, 3] \times [-3, 3]$ . We use the same level of precision for the solutions as



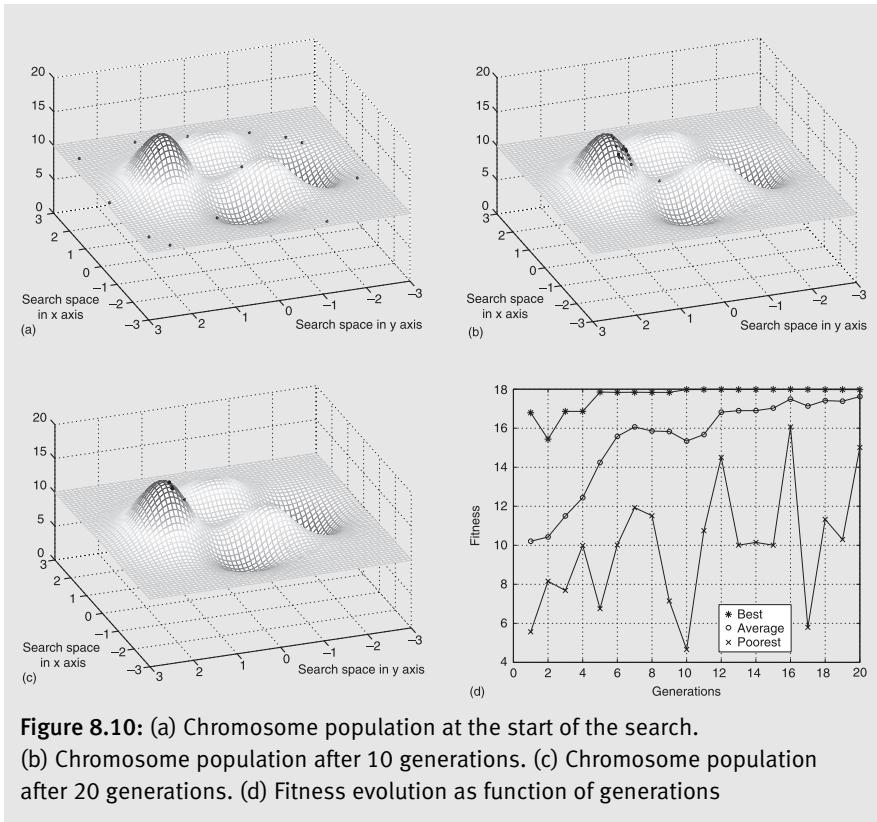
**Figure 8.9:** Chromosome concatenation in the case of a function of multiple variables

that used for the earlier example, i.e., three places of decimals. To make the function  $g$  take positive values within the allocated search space, the magnitude of the function is increased by a value of 10. All the steps involved for solving this problem are now similar to the ones used for solving the previous optimization example with the only modification made at the level of the length and structure of the chromosomes needed for encoding the search space.

In this example, the lowest  $n_i$  used to build the chromosome strings (for  $x$  and  $y$ ) is given by 13. This number is readily obtained by solving for the following inequality equation as outlined in section 8.5.4:  $((3 - (-3)) \cdot 10^3 \leq 2^{n_i} - 1$ . This is applied for the two variables  $x$  and  $y$ . The individuals (compounded chromosomes) used in the GA procedure are now composed of two concatenated sub-chromosomes, each composed of strings of 13 bits. To illustrate the idea, let us presume that two elements  $x$  and  $y$  in the search space grid are given as (1.353) and (-0.557), respectively. Their binary representation is then given as 1011100110101 and 0001001011100, respectively. As such, their concatenated chromosome representation composed of 26 bits (13 + 13) is given as in Figure 8.9.

Using the compounded chromosomes representation, all the steps outlined in the earlier example can be reproduced in the same way. Once the optimization is reached (we have chosen here the same parameters for crossover and mutation probability as in the previous example), the concatenated chromosome is then decoupled to give the corresponding values of  $x$  and  $y$ . In this particular example, the optimum of the peaks was reached at a value of  $g(x, y)$  given by 18.032 corresponding to  $x = -0.101$  and  $y = 1.602$  after 20 generations of a population of 20 individuals each. The search process is illustrated in Figures 8.10(a), (b) and (c) for the initial population, for a population evolved over 10 generations, and for a population evolved over 20 generations, respectively. Figure 8.10(d) represents the fitness evolution as a function of the number of generations for the poor, the average and the best outcome. Taking out the bias element of (+10) from the obtained result, the optimized value arrived at matches very closely the “correct” solution which is: (8.08).

All the concepts developed so far could be extended without loss of generality to functions of many variables.



**Figure 8.10:** (a) Chromosome population at the start of the search.  
 (b) Chromosome population after 10 generations. (c) Chromosome population after 20 generations. (d) Fitness evolution as function of generations

## 8.6 Integration of genetic algorithms with neural networks

Using GAs for the purpose of enhancing the learning capabilities of neural networks has been suggested since the early days of the backpropagation learning algorithm in the mid-1980s. But it was only recently that powerful and more formal algorithms have been developed for integrating the optimization tools of GAs with the learning schemes of a large class of neural networks. This integration has led to more powerful structures of ANN and helped enhance the conventional learning techniques. A number of researchers have since used genetic algorithms for a multitude of applications, namely for selecting appropriate inputs for neural networks and for improving the learning process of ANN, particularly those based on gradient descent procedures. These applications are briefly previewed next.

### 8.6.1 Use of GAs for ANN input selection

In [20], Guo and Uhrig used genetic algorithms to improve the application of neural networks for fault diagnosis in nuclear power plants. In their work, they used genetic algorithms to get the best combination of neural network

inputs from a large set of the plant variables. The aim of using genetic algorithms for the selection of neural networks input is to achieve faster training and improved accuracy with the minimum number of inputs. More details on this application can be found in [23].

### 8.6.2 Using GA for NN learning

This is among the most common applications of GA to artificial neural networks. Encoding schemes have been proposed in this regard for representing the network parameters such as the connection weights and the bias terms. Whitley and Hanson [22] were among the first to use genetic algorithms for searching the optimized weight parameters without relying on any gradient information. This process is done according to the following steps also described in more detail in [23].

*Step 1* Encode a complete set of weights in a given string with a well-defined fitness value

*Step 2* Start with a randomly selected population (for which the individuals are strings of weights and biases of the network ordered in a certain manner).

*Step 3* Proceed with the conventional GA operators to construct the child population from the parents' one.

The crossover operation in this process was found to have an important role in assembling “good” sub-blocks of strings to build a fitter population of children. One way of encoding the parameters of the networks is to order the weights and the biases in a sort of list structure as shown in [23] and as illustrated in Figure 8.11. The chromosome can then be expressed as:

$$(0.8, 0.5, -0.1, -0.1, \mathbf{2.3}, \mathbf{1.1}, 0.1, -1, 0.2, -1, 1.2, -0.8, \mathbf{-0.7}, 1.3)$$

where the bold-faced numbers represent the bias terms while all other numbers represent the weight values.

Once encoded in the chromosomes, training patterns are presented to the system and errors are computed. In this procedure, the fitness function is often

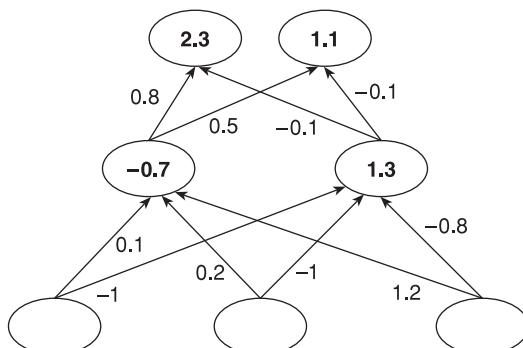


Figure 8.11: Encoding of a neural network

taken as the inverse of the network error scaled by the total sum error of the population. A combined scheme of genetic algorithms with neural networks (GANN) has the ability to locate the neighborhood of an optimal solution quicker than that found using conventional learning methods (such as backpropagation). This is due to the global search strategy of GAs. The only issue here is that once the chromosomes are in the optimal neighborhood, the GANN algorithm tends to converge more slowly to the optimal solution than is the case with the backpropagation learning rule. For this reason, several researchers have added another GA operator known as the gradient operator. This operator takes effect once the solution has reached the neighborhood of the global optimum.

## 8.7 Integration of genetic algorithms with fuzzy logic

In a similar manner to that done in the case of neuro-fuzzy modeling, fuzzy systems can be improved by providing means for optimizing their parameters, and GA could be used to improve this aspect. Fuzzy systems can model general nonlinear mappings in a manner similar to feedforward neural networks since it is a well-defined function mapping of real-valued inputs to real-valued outputs. All that is needed for practical application is a means for adjusting the system parameters so that the system output matches the training data. Genetic algorithms can provide such means. Table 8.5 illustrates the advantages and disadvantages of fuzzy systems and genetic algorithms.

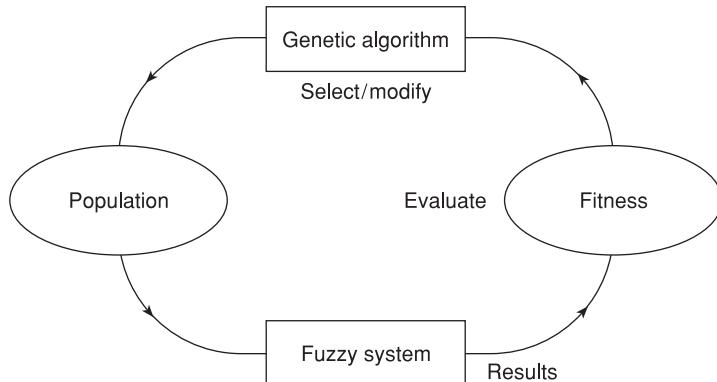
The integration of genetic algorithms with fuzzy systems has two main goals:

- (1) Improving the design process of fuzzy systems.
- (2) Improving the performance of fuzzy systems.

This performance can be interpreted as the accuracy of the control action and the efficiency (in terms of time computation). Genetic algorithms could be used to find optimized values for the membership function parameters, particularly when manual selection of their values becomes difficult or takes too much time to attain. The main issue here is to define an appropriate fitness function that serves as an adequate representative of the optimization process. Once the fuzzy rules have been set and the parameterized membership

**Table 8.5:** Advantages and drawbacks of fuzzy logic and GAs

Properties	Fuzzy systems	Genetic algorithm
Store knowledge	Explicit	None
Learns	No learning	Ability to learn
Optimizes	None	Powerful
Fast	Yes	Yes
Handle nonlinearity	Yes	Yes



**Figure 8.12:** Integrated scheme of GA and fuzzy logic systems

functions have been initialized, the GA module will then tune the values of the parameters subject to attaining the optimum of the fitness function. The procedure for using genetic algorithms to tune fuzzy systems is illustrated in Figure 8.12.

For example, a chromosome can be defined to be a concatenation of the values of all the membership functions. When triangular functions are used to represent the membership functions, for instance, the parameters are the centers and widths for each category. From an initial pool of possible parameter values, the fuzzy system is run to determine how well it performs. This information is used to determine the fitness of each chromosome and to establish the new population. The cycle is repeated until the best set of values for the membership function parameters is found.

## 8.8 Known issues in GAs

Despite several attractive features in terms of parallel search capabilities and abilities in dealing with discrete optimization problems, GAs have a number of drawbacks, as is the case for other optimization techniques. These are illustrated next.

### 8.8.1 Local minima and premature convergence

GAs are typically used on complex problem spaces that are difficult to understand or predict since they are effective at exploring such spaces. However, *local minima* (or maxima) are often found in complex spaces. These are regions of the space that hold good solutions relative to the regions around them, but which do not necessarily contain the best solutions in the problem space. The region(s) that contain the best solution(s) are called *global minima*. Purely exploitative algorithms, such as hill-climbers, may be trapped by local minima and never leave the region.

GAs are less prone to being trapped because of the effects of mutation and crossover. However, if a GA explores such a region extensively, it may be almost completely dominated by solutions within that region. A GA dominated by a set of identical or very similar solutions is often said to have *converged* (note, however, that there are many different definitions of convergence). Converged GAs are often very slow to escape a region of the problem space since novel solutions are overwhelmed by the dominant solutions. If a GA converges to a set of solutions within a local minimum, which is not a global minimum, then the GA is said to have *converged prematurely*, in the sense that it has converged before finding a global minimum. It is important to note that convergence itself is not a bad thing. Convergence to solutions in the global minimum is ideal and convergence of any kind is a useful means for observing the progress of a GA. Among the most significant factors contributing to convergence is the mutation rate. If it is too low, the population may converge rapidly to a local minimum it encounters and may never escape it.

### 8.8.2 Mutation interference

*Mutation interference* occurs when mutation rates in a GA are so high that solutions are so frequently or drastically mutated such that the algorithm never manages to explore any region of the space thoroughly. Even if it finds good solutions, they tend to be rapidly destroyed. This is the opposite problem to premature convergence. A GA experiencing mutation interference will probably never converge since its population is too unstable. Finding a mutation rate that allows the GA to converge but which also allows it to explore adequately is essential and often requires experimentation.

### 8.8.3 Deception

In some cases, a problem space may lead a GA to converge naturally to a sub-optimal solution, which may appear good. This is often the case when there are many decent solutions of similar form. It may also be that the region of attraction around the global minima is very small, and there exist other local minima with much larger regions of attraction. Thus, these minima mislead the GA as to the form of the best solution in a manner known as *deception*. This problem is very difficult to deal with since it may simply be inherent in the problem space itself. One technique for dealing with deception is to emphasize the difference between solutions by altering the objective function or the selection operator such that the best solutions are rated much higher than the deceiving ones.

### 8.8.4 Epistasis

*Epistasis* refers to a condition in the genotype structure where genes are highly interdependent. A set of genes may only produce a good solution when

their alleles occur in a particular pattern. It may prove difficult for a GA to discover good solutions since fragments of the patterns will have only a low value and thus do not provide information to guide the construction of better patterns. Again, the problem is likely to be inherent in the problem's structure and may be difficult to avoid. It may, however, be a feature of a particular genotype structure used to represent the problem. If the problem can be restated using another genotype structure and objective function, the epistasis may be avoided.

## 8.9 Population-based incremental learning

Population-based incremental learning (PBIL) [24] is an algorithm that integrates in an efficient way the features of genetic algorithms with those of competitive learning. This special combination results in a powerful tool that is considerably simpler than a GA, and exceeds the capabilities of a GA (in terms of speed and accuracy) on a large range of optimization problems. This section presents an overview of the PBIL algorithm.

### 8.9.1 Basics of PBIL

The PBIL algorithm aims at extracting the population statistics rather than maintaining a large number of samples. To achieve this goal, the algorithm initializes a probability vector that can play the role of a prototype for high evaluation solutions. Using the probability vector (the prototype), the population of the next generation can be generated, where the solution is encoded into a fixed-length vector. An evident feature of this mechanism is that it requires less memory and runs much faster than the conventional genetic algorithm. Another significant feature of using the probability vector is that the genetic operators take place directly on the vector rather than on individual solutions vectors.

### 8.9.2 Generating the population

PBIL algorithm utilizes the binary encoding scheme to create the probability vector. In this bit string presentation, the probability of each bit position containing a “1” is specified. The probability that a bit position contains a “0” can be determined by subtracting the probability in the vector from 1. Based on these probabilities, the members of a population can be extracted. It should be mentioned here that the diversity of the population depends on the probability values in the probability vector. A probability vector where the value in each bit position is assigned to 0.5 introduces the most diversity. In such a vector, the generation of 1 or 0 in each bit position is completely random. The following example shows the probability representation of three small populations where every solution in each population is a 4-bit vector, and the population size is 4.

### Example 8.4

#### *Probability vector representation*

This example illustrates the probability representation for four simple populations. The length of the chromosomes is 4. Notice (in Table 8.6) that the highest diversity occurs in populations 1 and 3. The lowest diversity is found in population 4.

**Table 8.6:** Probability vector representation of Example 8.4

Population 1	Population 2	Population 3	Population 4
1100	1000	0101	1111
1010	1100	1010	1110
0011	1100	0100	1110
0011	1110	1001	1111
<b>Representation</b>	<b>Representation</b>	<b>Representation</b>	<b>Representation</b>
0.5, 0.5, 0.5, 0.5	1.0, 0.75, 0.25, 0.0	0.5, 0.5, 0.5, 0.5	1.0, 1.0, 1.0, 0.5

The probability vector utilized by the PBIL is considered as a prototype vector for generating chromosomes that have high evaluation values. In each generation, the probability vector is updated in order to represent the fittest chromosome in the generation. This update mechanism is motivated by the competitive learning in neural networks. Similar to the training scheme of competitive learning, the probability vector values are updated with a view to representing solutions (chromosomes) with high evaluation values.

### 8.9.3 PBIL algorithm

The PBIL algorithm consists of the following steps [23]:

*Step 1* Initialize the probability vector,  $P$

*REPEAT*

*Step 2* Generate samples:

*Step 2.1* Generate sample vector according to probabilities in  $P$ .

*Step 2.2* Evaluate the sample vector.

*Step 3* Find the vector corresponding to maximum evaluation

(max ≡ best sample)

*Step 4* Update the probability vector according to the following rule:

$$P_i = P_i * (1 - LR) + \text{max}_i * (LR), \quad (8.12)$$

where  $LR$  represents the learning rate.

*Step 5* Mutate the probability vector.

if (random(0, 1) < Mutation Probability)

$$P_i = P_i * (1 - Mut\_Shift) + rand(0, 1] * (Mut\_Shift), \quad (8.13)$$

where *Mut\_Shift* is the amount of mutation affecting the probability vector.

PBIL explicitly maintains the statistics contained in a GA's population, without the use of genetic operators. PBIL is simpler than a GA, both theoretically and computationally. Empirical results show that PBIL is faster and more effective than standard GAs on a large set of benchmark problems. PBIL has been applied to several areas such as autonomous highway vehicle navigation, discrete point data selection for object localization, the design of low-level vision controllers for autonomous navigation, and the design of high-level reactive controllers for robot vehicles [25].

#### 8.9.4 PBIL and learning rate

The learning rate in the PBIL algorithm has a greater importance than in standard competitive learning. It directly affects how fast the probability vector gets altered to correctly resemble a classified point [24]. Since the population samples are generated based on the probability vector, the learning rate affects which portion of the search space is explored. This has a direct effect on the trade-off between *exploration* and *exploitation* of the search space. Exploration describes the ability of the algorithm to search the space thoroughly while exploitation depicts the ability of the algorithm to effectively use the gained information to reduce and narrow the future search. For example, if the learning rate is set as 0, there will be no exploitation of the information gained during the search. As it is increased, exploitation increases and the ability to search a large portion of the search space is reduced. In other words, the higher the learning rate, the faster the algorithm focuses the search. The lower the learning rate, the more exploitation takes place.

## 8.10 Evolutionary strategies

This variant of evolutionary computing was first suggested by Rechenberg [11] in the mid-1970s and was first applied for parameter optimization. Two main differences exist between evolution strategies (ES) and genetic algorithms (GA). They pertain essentially to the presentation schemes of individuals used in the search process and to the types of evolution operators applied. In GA implementations, the solutions are represented in the form of character strings where each digit of the string could take limited values (in most cases binary strings). However, in ES, real-valued strings are used to represent the solutions. Also, while GAs incorporate both crossover and mutation operations, ESs use only the mutation operator. At first glance, it may

seem that ES are easier to implement and may require fewer computational resources than their GA counterparts to reach the optimum solution. However, this is not always the case and the *no free lunch theorem* [27] states that there doesn't exist a universal "Best Optimization Algorithm". As such each algorithm serves efficiently a specific application domain.

The basic implementation of evolution strategies is known as the *two member*  $(1+1)$  – ES scheme. This means that one parent generates one offspring and the best of the two is selected for further processing while the other is eliminated from the sample set. In this section, we highlight this basic form which illustrates the key concepts of ES. We then introduce a few extensions to the basic form.

Presuming the objective is to find a set of values for a set of parameters to satisfy certain constraints while optimizing an objective (or fitness) function, this type of problem could be solved by ES in the following steps:

*Step 1* Find a real-valued vector  $X = (x_1, x_2, \dots, x_m)$ , which represents the original set of parameters. We should be able to compute the original set of parameters given the presentation vector and vice versa.

*Step 2* Create a new offspring  $X'$  by mutation. To achieve the mutation, add a random vector of the same size as  $X$  with normal distribution  $N(0, \sigma)$  (zero mean and standard deviation  $\sigma$ ). The new offspring has its elements given by:

$$x'_i = x_i + N(0, \sigma_i) \quad (8.14)$$

From the mathematical analysis for two sample cost functions, Rechenberg [11] suggested the following heuristic strategy for adjusting  $\sigma$  (step size) through the course of iterations. It is known as the *1/5 success rule*. It states that the ratio of successful mutation (where the child is fitter than the parent) to all mutations should be  $1/5$ . If the ratio is greater than  $1/5$ , then increase the standard deviation to widen the search of the space; if it is less, then decrease the mutation standard deviation to focus the search. This translates to the following mathematical rule pertaining to the standard deviation (step size):

$$\sigma = \begin{cases} \sigma/c & \text{if } p_s > (1/5) \\ \sigma c & \text{if } p_s < (1/5) \\ \sigma & \text{if } p_s = (1/5) \end{cases} \quad (8.15)$$

where  $p_s$  is the mutation ratio and  $c$  is a positive parameter belonging to the interval  $[0.817 \quad 1.0]$  as suggested in [28].

*Step 3* Compare the solutions for  $X$  and  $X'$ . Choose the best member for the next generation.

*Step 4* Repeat steps 2 and 3 until a satisfactory solution is found or until the computation time is exhausted.

It was proven in [28] that for regular optimization problems, this algorithm converges to global optimum with probability 1:

**Theorem:** For a regular optimization problem with cost function  $f$  and global optimum  $f^* > -\infty$ :

$$\text{Prob} \left\{ \lim_{t \rightarrow \infty} f(x^t) = f^* \right\} = 1 \quad (8.16)$$

The reader may consult [28] for a proof of this theorem. Notice that for the two member-scheme  $(1 + 1) - ES$ , we don't use the concept of population in search, and hence it could be considered as a "point to point" search. Therefore it can likely be trapped in local maxima (although it converges to global maximum with probability 1). In order to improve the algorithm and decrease entrapment risk, Rechenberg [11] suggested two new extensions to the basic ES: the  $(\mu + \lambda) - ES$  and the  $(\mu, \lambda) - ES$  schemes, also known as the plus strategy and the comma strategy, respectively. For both schemes,  $\mu$  represents the number of parent individuals and  $\lambda$  represents the number of offspring (children) individuals. The other suggested improvement was to change mutation variance adaptively from population to population. In other words, we impart some sort of learning to the search algorithm. The modified ES algorithms could be summarized as follows:

*Step 1* Choose  $\mu$  parent vectors that contain  $m$  parameters  $X = (x_1, x_2, \dots, x_m)$ . Each parameter is chosen through a random process and satisfies the constraints of the problem.

*Step 2* Create  $\lambda$  new offspring ( $\mu \prec \lambda$ ) by recombining  $\mu$  parents with mutation like step 2 in the  $(1 + 1) - ES$  scheme. There are five types of recombination operators:

- (1) No recombination: Select one parent randomly and let  $x_i'' = x_i$ .
- (2) Discrete: Select two parents  $a$  and  $b$  randomly and let  $x_i' = x_{i,a}$  or  $x_i' = x_{i,b}$  with equal probability.
- (3) Intermediate: Select two parents  $a$  and  $b$  randomly and let  $x_i' = \frac{1}{2}(x_{i,a} + x_{i,b})$ .
- (4) Global Discrete: Select a new pair of  $a_i$  and  $b_i$  parents for every parameter  $x_i$  and let  $x_i' = (x_{a_i,1} \text{ or } x_{b_i,2})$  with equal probability.
- (5) Global Intermediate: Select a new pair of  $a_i$  and  $b_i$  parents for every parameter  $x_i$  and let  $x_i' = \frac{1}{2}(x_{a_i,1} + x_{b_i,2})$ .

Generate the offspring population with the following procedure:

Current population:  $P^t$

Intermediate offspring:  $X'(x_1', x_2', \dots, x_m')$

Mutation operator:  $M$

Step size meta-control:  $\Delta\sigma$

Recombination operator:  $R$

$$(X', \sigma') = R(P^t) \quad (8.17)$$

$$(X'', \sigma'') = M[(X', \sigma')] \quad (8.18)$$

$$\sigma'' = \sigma' \cdot e^{N(0, \Delta\sigma)} \quad (8.19)$$

$$X'' = X' + N(0, \sigma'') \quad (8.20)$$

Note that the mutation operator is applied for both parameters and correspondent variance.

*Step 3* Select the most fit solutions  $\mu$  for the next generation:

For the plus scheme  $(\mu + \lambda) - ES$ , select the individuals of the next generation from the population of all parents and offspring  $(\mu + \lambda)$ . For the comma scheme  $(\mu, \lambda) - ES$ , select the next generation from the  $\lambda$  population of offspring, which means that the best offspring replace parent individuals.

*Step 4* Repeat steps 2 through 3 until a satisfactory solution is found or the computation time is exhausted.

To illustrate the behavior of a simple implementation of ES, we solve a simple optimization problem in the following example.

### Example 8.5

We wish to find here the maximum of the mapping to two variables given by

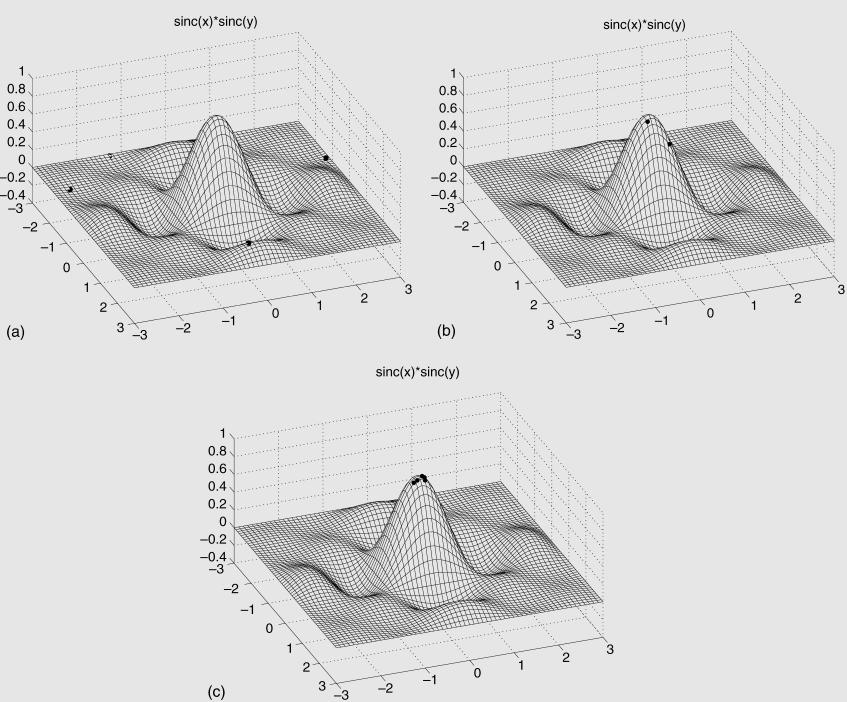
$$F(x, y) = \sin c(x) \cdot \sin c(y)$$

where

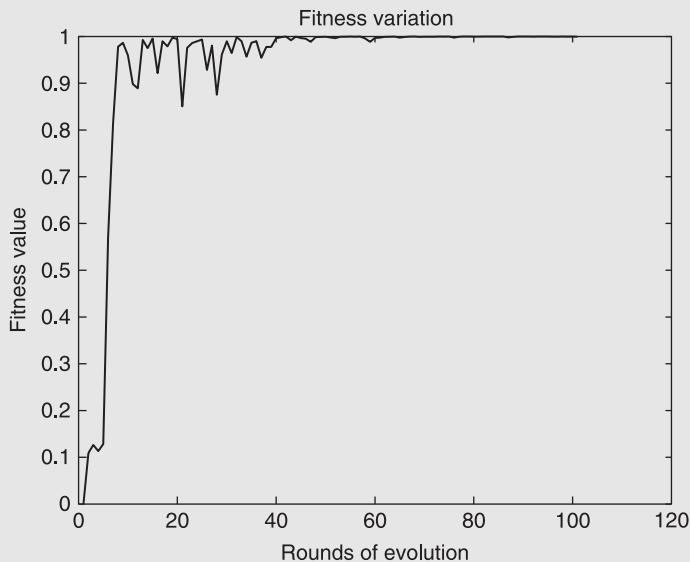
$$\sin c(x) = \begin{cases} \frac{\sin(\pi x)}{x} & \text{if } x \neq 0 \\ 1 & \text{if } x = 0 \end{cases}$$

To solve this optimization problem using ES, we implement a simple  $(3 + 5) - ES$  scheme which comprises only mutation with a fixed evolution strategy parameter of  $\sigma = 0.1$ . Figure 8.13(a) shows the objective function along with the location of the first generation of individuals.

For our implementation of ES, we consider a simple ES which comprises only mutation with a fixed evolution strategy parameter of  $\sigma = 0.1$ . After 10 rounds of evolution the location of the new sample population is depicted in Figure 8.13(b). One can easily notice that the position of the new individuals has generally moved towards the global maximum. After 30 rounds of evolutions and as shown in Figure 8.13(c), several individuals of the sample population have migrated very close to the global maximum of the objective function, which is given theoretically by the value 1. Figure 8.14 shows the behavior of the fitness (objective) function as a function of the number of rounds of evolution.



**Figure 8.13:** (a) Evolution of the population of individuals after one round of evolution, (b) after 10 rounds of evolution and (c) after 30 rounds of evolution



**Figure 8.14:** Behavior of the objective function as the number of evolution rounds increases

## 8.11 ES applications

Evolution strategies look promising in terms of optimization tools that have been applied successfully to a number of real engineering applications. In the following section we highlight recent applications of ES in different disciplines.

### 8.11.1 Parameter estimation

ES – as a stochastic search – can be applied to system parameter estimation. In [29], Hatanaka *et al.* applied ES for multiple estimates of the system parameters, and illustrated their findings with numerical examples [29]. For system parameter estimation, adaptability and robustness are important factors. Adaptability is adaptiveness to the system dynamics and robustness pertains to robustness to outliers. Hatanaka *et al.* applied ES to the parameter estimation of the autoregressive (AR) model, and they used  $(\mu + \lambda)$  – ES selection. Finally, they showed the out-performance of ES over recursive least square and recursive weighted least square methods. They emphasized the adaptability and robustness of ES over other methods.

### 8.11.2 Image processing and computer vision systems

ES can also be used for image analysis applications. Louchet applied ES to stereo image analysis [30]. Because of the usually large image data size, ES-based image analysis suffers from heavy computation complexity. In [30], Louchet split the problem into several independent and simpler primitives. He used 3-D points, which is one of the simplest primitives. The main idea of Louchet is to evolve a population of 3-D points using a fitness function. The fitness function evaluates the similarity of the 3-D points of each stereo image. He then used a deterministic selection operator: a ranking process based on fitness values. He also claims real-time properties of ES because of their adaptive nature, i.e., coping with modifications of the fitness function during the algorithm's run.

### 8.11.3 Task scheduling by ES

Greenwood *et al.* applied ES for task scheduling in multiprocessor systems [31]. They illustrated the scheduling of a digital signal-processing algorithm on a two-processor distributed system. Multiprocessor scheduling is to assign a set of tasks for a multiprocessor system to minimize overall scheduling length, and this is one of the NP-complete problems. ES provided shorter scheduling time than other methods did.

### 8.11.4 Mobile manipulator path planning by ES

Watanabe *et al.* applied ES for omni-directional mobile manipulator path planning [32]. For the *B-spline*, the choice of the appropriate data points and

end points is most important. Thus, Watanabe *et al.* suggested automatic selection of those points using various cost functions: motion smoothness, movable range of joint, singular operation, and falling down. This path planning method is also useful for path generation with time constraints.

### 8.11.5 Car automation using ES

Ostertag *et al.* applied ES for airbag release optimization [33]. They presented a tuning method for airbag release. A quality function and a modified ES are introduced. Airbag release optimization is a difficult problem because there are many different crash situations and airbag misfiring is dangerous and leads to high repair costs. The quality function is defined for optimal performance. However, it includes erroneous trigger decisions and timings. Because of the characteristics of the quality function, it is difficult to apply standard methods such as gradient descent or hill-climbing. In most of their experimental testing, they were able to obtain solutions very close to the optimal solution.

## 8.12 Summary

Although much has been achieved in recent years in terms of investigating global optimization problems using genetic algorithms and other evolutionary algorithms, computational resource requirements have been among the major hindering aspects, especially if an on line solution is required. Genetic algorithms, while possibly allowing for the global optima of a given function, rarely reach the exact optima. This is a shortcoming of the technique, particularly if an exact solution is required. Moreover, it is quite difficult in some instances (due to possible limitations of computational resources) to implement a given GA in real time. Advances have been made in this regard by exploiting the parallel capabilities of the GA operators and implementing them on distributed processors. Several researchers have also investigated in recent years the implementation of hybrid techniques in which a genetic algorithm provides for the approximate location of the optima, while other gradient-descent-based techniques locate the exact location of these optima using derivative-based techniques. A lot of research has been carried out in this direction to optimize computational resources and allow these techniques to solve an even wider range of optimization problems. The PBIL-based optimization technique and evolutionary strategies have been typical examples of such research activity.

### Problems

1. Enumerate the advantages and disadvantages of the three major optimization algorithms: gradient descent technique, Newton-based technique, and genetic algorithms. Derive some conclusion about the single-based and population-based optimization methods.

2. Discuss how the “learning rate” affects the performance of the population-based learning algorithm.

3. Solve the optimization of the function mentioned in example 8.1:

$$f(x) = 20 + 100 * \cos(4\pi x) \exp(-2x)$$

- (a) by using the following values for crossover rate and mutation rate:

$$(0.1, 0.01), (0.01, 1), (1, 0.01), (0.8, 0.8), (0.1, 0.8)$$

- (b) Interpret the results obtained in terms of accuracy and computation time required. Go over the same problem by fixing the rate at (0.5, 0.01) and varying the population size: 10, 15, 20, and 40.

4. The Rosenbrock Banana function is defined as follows:

$$f(x_i) = 100 * (x_{i+1} - x_i^2)^2 + (1 - x_i)^2.$$

Minimize this function over the interval [-2, 2] with an accuracy of two decimals.

5. Using GA, solve the optimization of the following function:

$$f(\vec{x}) = \sum_{i=1}^n |x(i)|^{(i+1)},$$

where all  $n$ -component  $x(i)$  of the vector  $\vec{x}$  belong to the interval [-1, 1]. Make the optimization for  $n = 2, 3, 4$ , with an accuracy of three decimals.

6. Optimize the Schwefel function given by

$$f(\vec{x}) = \sum_{i=1}^n \left( \sum_{j=1}^i x(j)^2 \right)^2$$

where all  $n$ -component  $x(i)$  of the vector  $\vec{x}$  belong to the interval [-65.536, 65.536]. Make the optimization for  $n = 2, 3$ , and 4 with an accuracy of three decimals.

7. Repeat problems 3, 4, and 5 using evolutionary strategies. Comment on the outcome of the results in terms of accuracy or computational requirements.

8. Neural networks and genetic algorithms are both soft computing techniques used for learning and optimization. While neural networks use inductive learning and rely on training data, GAs use deductive learning and require performance indices for objective evaluation. Discuss these two soft computing tools and infer that both techniques could integrate to generate a better overall performance system known as an evolutionary neural network.

9. We need to approximate the mapping of problem 6 in Chapter 5 using a feed-forward neural network combined with a GA-based optimization. Using a representation of the chromosomes similar to the one given in 8.6.2 and using the same architecture as in problem 6 in Chapter 5, compare the results with those obtained using the backpropagation learning algorithm (based on the gradient descent approach).

## References

1. Deb, K. (1996) “Genetic algorithms for function optimization,” in F. Herrera and J. Verdegay (eds), *Genetic Algorithms and Soft Computing*, pp. 3–29. Physica-Verlag.

2. Jang, J.S.R., Sun, C.T., and Mizutani, E. (1997) *Neuro-Fuzzy and Soft Computing*, Prentice Hall, Englewood Cliffs.
3. Pedrycz, W. (1998) *Computational Intelligence: an introduction*, CRC Press, New York.
4. Tsoukalas, L., and Uhrig, R. (1997) *Fuzzy and Neural Approaches in Engineering*, John Wiley & Sons, New York.
5. Michalewicz, Z., and Michalewicz, M. (1997) "Evolution computation techniques and their application," in *IEEE International Conference on Intelligent Processing Systems*, pp. 14–26.
6. Michalewicz, Z. (1994) *GA + Data Structure = Evolution Programming*, Springer-Verlag, New York.
7. Ross, P., and Corne, D. (1995) *Application of Genetic Algorithms*, Prentice Hall PTR, Englewood Cliffs.
8. Back, T., Hammel, U., and Schwefel, H.P. (1997) "Evolutionary computation: comments on the history and current state," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 3–17.
9. Fogel, D. (1994) "An introduction to simulated evolutionary optimization," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 3–14.
10. Fogel, L., Owens, A., and Walsh, M. (1966) *Artificial Intelligence through Simulated Evolution*, John Wiley, Chichester.
11. Rechenberg I. (1973) *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*, Frommann-Holzboog, Stuttgart.
12. Schewfel, H. (1981) *Numerical Optimization of Computer Models*. John Wiley & Sons, New York.
13. Baeck, T., Hoffmeister, F., and Schwefel, H.P. (1991) "A survey of evolution strategies," in *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 2–9.
14. Langdon, W. (1994) *GP + Data structure = Automatic Programming*, Kluwer Academic Publisher, New York.
15. Koza, J. (1992) *Genetic Programming: On the Programming of Computers by means of Natural Selection*, MIT Press.
16. Holland, J. (1975) *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor.
17. Holland, J., and Reitman, J. (1978) "Cognitive systems based on adaptive algorithms," in D. Waterman and F. Hayes (eds), *Pattern Directed Inference Systems*, Academic Press, New York.
18. Wienholt, W. (1993) "A refined genetic algorithm for parameter optimization problems," *Proceedings of the Fifth International Conference on Genetic Algorithms*, pp. 589–96, Morgan Kaufmann Publishers.
19. Michalewicz, Z. (1994) *Genetic Algorithms + Data Structures = Evolution Programs*, Springer Verlag, Berlin.
20. Guo, Z., and Uhrig, R.E. (1992) "Sensitivity analysis and applications to nuclear power plant," *International Joint Conference on Neural Networks*, vol. 2, pp. 453–58.
21. Koza, J. (1992) *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press.
22. Whitley, D. (1989) {GENITOR} "Algorithm and selection pressure: why rank-based allocation of reproductive trials is best," *Proceedings of the Third International Conference on Genetic Algorithms*, [ICGA3], pp. 116–21, San Mateo, California.
23. Lin, C.T., and Lee, C.S.G. (1996) *Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems*, Prentice Hall, Englewood Cliffs.

24. Baluja, S. (1994) *Population-based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning*, Technical Report CMU-CS-94-163, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA.
25. Sukthankar, R., Baluja, S., and Hancock, J. (1997) "Evolving an intelligent vehicle for tactical reasoning in traffic," *Proceedings of 1997 IEEE International Conference on Robotics and Automation*, vol. 1, pp. 519–24.
26. Holland, J. (1975) *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, USA.
27. Fogel, D. (2002) *Evolutionary Computing*, IEEE Press.
28. Schwefel, H.P. (1995) *Evolution and Optimum Searching*, Wiley Interscience, John Wiley & Sons, New York.
29. Hatanaka, T., Uosaki, K., Tanaka, H. and Yamada, Y. (1996) "System parameter estimation by evolutionary strategy," *Proceedings of the 35th SICE Annual Conference*, pp. 1045–48.
30. Louchet, J. (2000) "Stereo analysis using individual evolution strategy," *Proceedings of the 15th International Conference on Pattern Recognition*, vol. 1, pp. 908–11.
31. Greenwood, G.W., Gupta, A., and McSweeney, K. (1994) "Scheduling tasks in multiprocessor systems using evolutionary strategies," *Proceedings of the First IEEE Conference on Evolutionary Computation*, vol. 1, pp. 345–9.
32. Watanabe, K., Kiguchi, K., Izumi, K., and Kunitake, Y. (1999) "Path planning for an omnidirectional mobile manipulator by evolutionary computation," *Third International Conference on Knowledge-Based Intelligent Information Engineering Systems, 1999*, pp. 135–40.
33. Ostertag, M., Nock, E., and Kiencke, U. (1995) "Optimization of airbag release algorithms using evolutionary strategies," *Proceedings of the 4th Conference on IEEE Control Applications*.

# Applications and case studies



# Soft computing for smart machine design

## Chapter outline

9.1	Introduction	407
9.2	Controller tuning	413
9.3	Supervisory control of a fish processing machine	427
9.4	Summary	467
	Problems	467
	References	472

## 9.1 Introduction

Smart machines will exhibit an increased presence and significance in a wide variety of applications. Process industries have a significant potential for using intelligent machines, incorporating advanced sensor technology and intelligent control. Tasks involved may include handling, cleaning, machining, joining, assembly, inspection, repair, and packaging. In industrial plants many of these tasks are still not automated, and use human labor. It is important that intelligent machines perform their tasks with minimal intervention of humans, maintain consistency and repeatability of operation, and cope with disturbances and unexpected variations in the machine, its operating environment, and performance objectives. In essence, these machines should be autonomous and should have the capability to accommodate rapid reconfiguration and adaptation. For example, a production machine should be able to quickly cope with variations ranging from design changes for an existing product to the introduction of an entirely new product line. The required flexibility and autonomous operation translate into a need for a higher degree of intelligence in the supporting devices. This will require proper integration of such devices as sensors, actuators, and controllers, which themselves may have to be “intelligent” and, furthermore, appropriately distributed throughout the system. Design, development, production, and operation of intelligent machines, which integrate technologies of sensing,

actuation, and intelligent control, have been made possible today through ongoing research and development in the field of intelligent systems and control.

### 9.1.1 Intelligent machines

In broad terms, an intelligent machine may be viewed as consisting of a *knowledge system* and a *structural system*. The knowledge system effects and manages intelligent behavior of the machine. The physical elements of a machine are not intelligent but the machine can be programmed to behave in an intelligent manner. The structural system consists of physical hardware and devices that are necessary to perform the machine objectives yet do not necessarily need a knowledge system for their individual functions. Sensors, actuators, controllers (nonintelligent), communication interfaces, mechanical devices, and other physical components fall into this category, and they will work cooperatively in generating an intelligent behavior by the machine. Sensing with an understanding or “feeling” of what is sensed is known as *sensory perception*, and this is very important for intelligent behavior. For proper functioning of an intelligent machine it should have effective communication links between the various components, and an interface to the external world. This broad division of the structure of an intelligent machine is primarily functional rather than physical. In particular, the knowledge system may be distributed throughout the machine, and individual components themselves may be interpreted as being “intelligent” as well (for example, intelligent sensors, intelligent controllers, intelligent multi-agent systems, and so on). An actual implementation of an intelligent system will be domain specific, and much more detail than what is mentioned here may have to be incorporated into the system structure. Even from the viewpoint of system efficiency, domain-specific and special-purpose implementations are preferred over general-purpose systems.

Advances in digital electronics, technologies of semiconductor processing, and micro-electromechanical systems (MEMS) have set the stage for the integration of intelligence into sensors, actuators, and controllers. The physical segregation between these devices may well be lost in due time as it becomes possible to perform diversified functionalities such as sensing, conditioning (filtering, amplification, processing, modification, etc.), transmission of signals, and intelligent control all within the same physical device. Due to the absence of adequate analytical models, sensing assumes an increased importance in the operation and control of complex systems such as intelligent machines. The trend in the applications of intelligent machines has been towards mechatronic-type technology where intelligence is embedded at the component level, particularly in sensors and actuators, and distributed throughout the system.

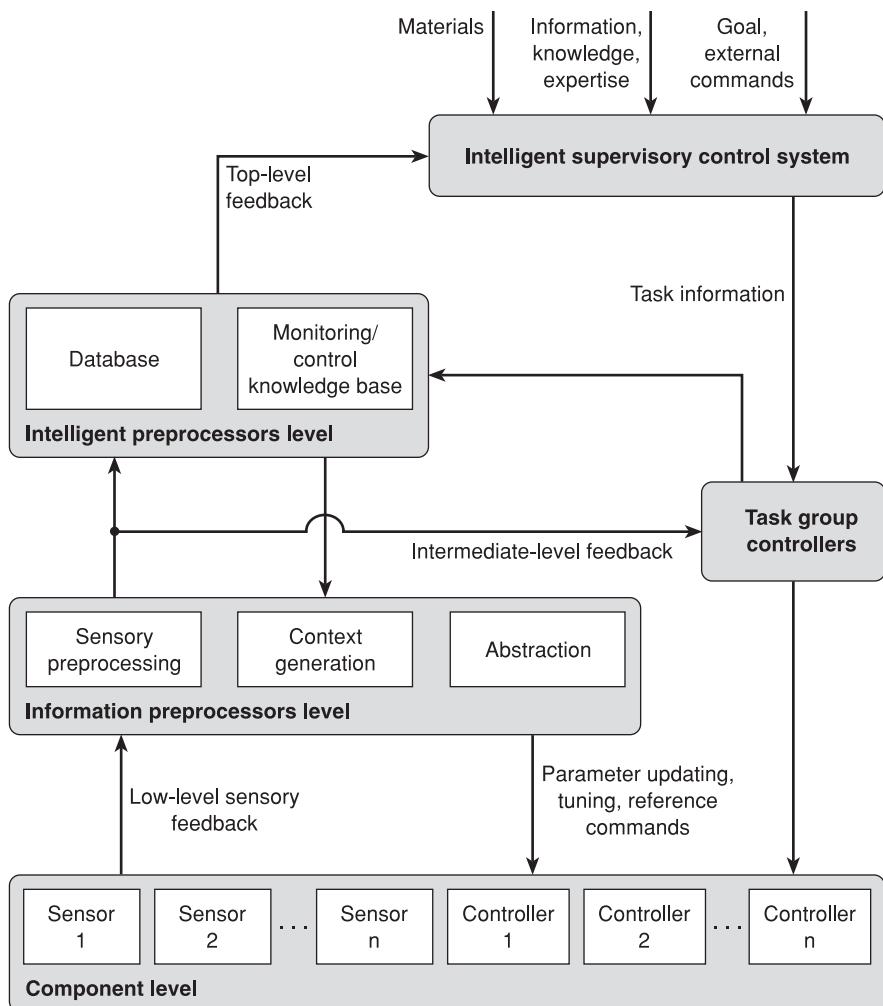
### 9.1.2 Intelligent control

Intelligent machines rely heavily on intelligent control in their operation. The term *intelligent control* may be loosely used to denote a control technique

that can be carried out using the “intelligence” of a human who is knowledgeable in the particular domain of control. In this definition, constraints pertaining to limitations of sensory and actuation capabilities and information processing speeds of humans are not considered. It follows that if a human in the control loop can properly control a machine, then that machine would be a good candidate for intelligent control. The need for intelligent control is evident from the emergence of engineering systems exhibiting an increasingly greater degree of complexity and sophistication. When the process is complex and incompletely known, yet some human expertise or experience is available on a practical scheme to control the process, the techniques of intelligent control may be particularly appealing. The field of intelligent control has grown in recent years at an impressive rate. This is due primarily to the proven capabilities of the algorithms and techniques, particularly with soft computing using fuzzy logic, developed in the field to deal with complex systems that operate in ill-defined and dynamically varying environments. Techniques of intelligent control are useful in autonomous control and supervision of complex machinery and processes. Intelligent machines are called upon to perform complex tasks with high accuracy, under ill-defined conditions. Conventional control techniques may not be quite effective under these conditions, whereas intelligent control has a tremendous potential. Unlike conventional control, intelligent control techniques possess capabilities for effectively dealing with incomplete information concerning the machine and its environment, and unexpected or unfamiliar conditions. Information abstraction and knowledge-based decision-making, which incorporates abstracted information, are considered important in intelligent control. Techniques of fuzzy logic are widely applied in intelligent control. There are numerous practical implementations of fuzzy control. In Japan, for example, consumer products and utilities such as washing machines, vacuum cleaners, hand-jitter-compensated video cameras, rice cookers, automobiles, and subway trains that use this method of control are already commercially available.

### 9.1.3 Hierarchical architecture

A hierarchical structure can facilitate efficient control and communication in an intelligent machine. A three-level hierarchy is shown in Figure 9.1. The bottom level consists of machine components with component-level sensing and control. Machine actuation and direct feedback control are carried out at this level. The intermediate level uses intelligent preprocessors for abstraction of the information generated by the component-level sensors. The sensors and their intelligent preprocessors together perform tasks of intelligent sensing. The state of performance of the machine components may be evaluated by this means, and component tuning and component-group control may be carried out as a result. The top level of the hierarchy performs task-level activities including planning, scheduling, machine performance monitoring, and overall supervisory control. Resources such as materials and expertise may be provided at this level and a human-machine interface would be available.



**Figure 9.1:** A hierarchical control/communications structure for an intelligent machine

Knowledge-based decision-making is carried out at both intermediate and top levels, and techniques of soft computing would be applicable in these knowledge-based activities. The resolution of the information that is involved will generally decrease as the hierarchical level increases, while the level of “intelligence” that would be needed in decision-making will increase.

Within the overall system, the communication protocol provides a standard interface between various components such as sensors, actuators, signal conditioners, and controllers, and also with the machine environment. The protocol will not only allow highly flexible implementations, but will also enable the system to use distributed intelligence to perform preprocessing and understanding of information. The communication protocol should be based

on an application-level standard. In essence, it should outline what components can communicate with each other and with the environment, without defining the physical data link and network levels. The communication protocol should allow for different component types and different data abstractions to be interchanged within the same framework. It should also allow for information from geographically removed locations to be communicated to the control and communication system of the machine.

#### 9.1.4 Development steps

Development of an intelligent machine will require a parallel development of the knowledge system and the structural system of the machine. It may be the case that the structural system (a nonintelligent machine) is already available. Still, modifications might be necessary in the structural system in order to accommodate the needs of the intelligent system, for example, new sensors, transducers, and communication links for information acquisition. In any event, the development of an intelligent machine is typically a multidisciplinary task often involving the collaboration of a group of professionals such as engineers (electrical, mechanical, etc.), domain experts, computer scientists, programmers, software engineers, and technicians. The success of a project of this nature will depend on proper planning of the necessary activities. The involved tasks will be domain specific and depend on many factors, particularly the objectives and specifications of the machine itself. The main steps would be:

- (1) Conceptual development
- (2) System design
- (3) Prototyping
- (4) Testing and refinement
- (5) Technology transfer and commercialization.

It should be noted that generally these are not sequential and independent activities; furthermore, several iterations of multiple steps may be required before satisfactory completion of any single step.

Conceptual development will usually evolve from a specific application need. A general concept needs to be expanded into an implementation model of some detail. A preliminary study of feasibility, costs, and benefits should be made at this stage. The interdisciplinary groups that would form the project team should be actively consulted and their input should be incorporated into the process of concept development. Major obstacles and criticism that may arise from both prospective developers and users of the technology should be seriously addressed in this stage. The prospects of abandoning the project altogether due to valid reasons such as infeasibility, time constraints, and cost–benefit factors should not be overlooked.

Once a satisfactory conceptual model has been developed, and the system goals are found to be realistic and feasible, the next logical step of

development would be system design. Here the conceptual model is refined and sufficient practical details for implementation of the machine are identified. The structural design may follow traditional practices. Commercially available components and tools should be identified. In the context of an intelligent machine, careful attention needs to be given to design of the knowledge system and the human-machine interface. System architecture, types of knowledge that are required, appropriate techniques of knowledge representation, reasoning strategies, and related tools of knowledge engineering should be identified at this stage. Considerations in the context of user interface would include graphic displays; interactive use; input types including visual, vocal, and other sensory inputs; voice recognition; and natural language processing. These considerations will be application specific to a large degree and will depend on what technologies are available and feasible. A detailed design of the overall system should be finalized after obtaining input from the entire project team, and the cost-benefit analysis should be refined. At this stage the financial sponsors and managers of the project as well as the developers and users of the technology should be convinced of the project outcomes.

Prototype development may be carried out in two stages. First a research prototype may be developed in the laboratory, for proof of concept. For this prototype it is not necessary to adhere strictly to industry standards and specifications. Through the experience gained in this manner a practical prototype should be developed next, in close collaboration with industrial users. Actual operating conditions and performance specifications should be carefully taken into account for this prototype. Standard components and tools should be used whenever possible.

Testing of the practical prototype should be done under normal operating conditions and preferably at an actual industrial site. During this exercise, prospective operators and users of the machine should be involved cooperatively with the project team. This should be used as a good opportunity to educate, train, and generally prepare the users and operators for the new technology. Unnecessary fears and prejudices can kill a practical implementation of advanced technology. Familiarization with the technology is the most effective way to overcome such difficulties. The shortcomings of the machine should be identified through thorough testing and performance evaluation where all interested parties should be involved. The machine should be refined (and possibly redesigned) to overcome any shortcomings.

Once the developed system is tested, refined, and confirmed to satisfy the required performance specifications, the processes of technology transfer to industry and commercialization could begin. An approved business plan, necessary infrastructure, and funds should be in place for commercial development and marketing. According to the existing practice, engineers, scientists, and technicians provide minimal input into these activities. This situation is not desirable and needs to be greatly improved. The lawyers, financiers, and business managers should work closely with the technical team during the processes of production planning and commercialization. Typically the industrial sponsors of the project will have the right of first refusal of the developed technology. The process of technology transfer

would normally begin during the stage of prototype testing and continue through commercial development and marketing. A suitable plan and infrastructure for product maintenance and upgrading are musts for sustaining any commercial product.

## 9.2 Controller tuning

In knowledge-based control, the control signals are generated by an appropriate inference mechanism from a control knowledge base, which is typically expressed as a set of rules. Conventional fuzzy control, in particular, uses linguistic rules of the typical form, “If the error is positive and large and the error rate is negative and small then slightly increase the control signal.” To come up with such rules the control engineer has to have either a thorough insight of the dynamics of the overall control system including the plant, or tremendous experience in actually applying various types of control signals to the system and observing the system response. Since knowledge-based control is intended for a complex or poorly-known plant it may not be realistic to assume that the dynamic behavior of the system is completely known. For a moderate or high-bandwidth process such as machine tools (here the term bandwidth refers to the speed or maximum frequency of response) it is also unrealistic to manually sequence the control signals and observe the system behavior on line, in order to acquire the necessary knowledge and experience for establishing a set of fuzzy control rules. On the other hand it is much easier for a human expert to manually tune the control system of moderate-to-high bandwidth during operation, and thereby gather “tuning knowledge.” Manual tuning is still widely used in process control practice.

Ziegler–Nichols type empirical and heuristic tuning rules are known to be quite effective for a variety of servo systems, and as a result many commercial systems of servo tuning based on the Ziegler–Nichols method are still available. Much of the practical experience in tuning a servo controller is available not as hard algorithms based on accurate models and rigorous analysis, but rather in the form of “soft” rules. These reasons and facts have led us to consider a particular type of control structure in which fuzzy logic is used not directly in control but rather at a higher level, for tuning the direct controllers that are located at a lower level. In this structure, crisp control algorithms or high-bandwidth servo controllers (here bandwidth refers to the frequency of control cycles) serve as a direct controller and a knowledge-based tuner of relatively low bandwidth occupies a higher level to monitor and tune the direct controllers. This structure naturally conforms to the fact that experience-based tuning protocols are inherently fuzzy for the most part. It has another main advantage in that fuzzy schemes are relegated to a higher level so that they do not enter into the direct control loops, thereby maintaining a high control bandwidth (or speed of control), perhaps improving the accuracy of control as well. It should be emphasized that our purpose here is to automate the human expertise of the control engineer, who is not the plant operator except in special circumstances.

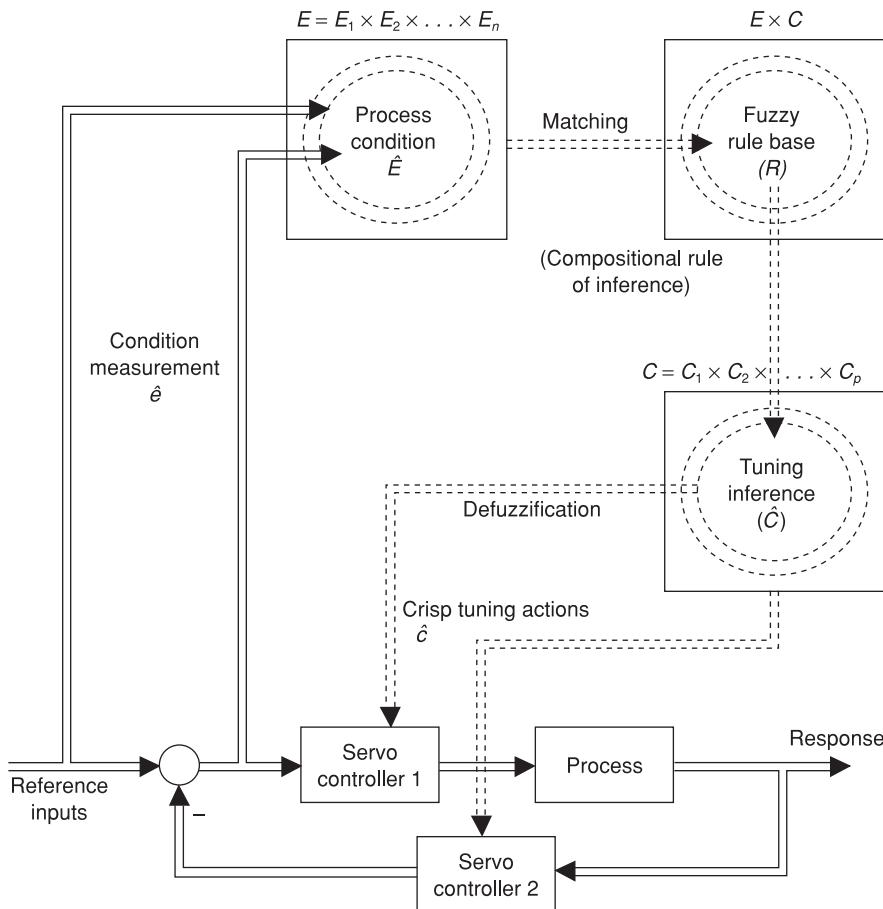


Figure 9.2: Structure of a knowledge-based servo tuner

In this section, the problem of tuning a servo controller, where a set of linguistic tuning protocols is available as the knowledge base, is considered. Fuzzy logic is used to represent the tuning knowledge base in the form of a membership function of a fuzzy relation, and this is extended to the tuning problem through the application of the compositional rule of inference. An example of tuning a proportional-integral-derivative (PID) servo for a non-minimum-phase process with transport delay is presented to illustrate the application of the concepts introduced here.

### 9.2.1 Problem formulation

Consider the general structure of a fuzzy tuner as shown in Figure 9.2. The control system is represented by the state-space error model

$$\dot{e}(t) = f(e(t), c(t), u(t)) \quad (9.1)$$

where time  $t \in \mathbb{R}^+$ ; the error state vector  $e(t): \mathbb{R}^+ \rightarrow \mathbb{R}^n$ ; the tuning parameter vector  $c(t): \mathbb{R}^+ \rightarrow \mathbb{R}^p$ ; the reference input vector  $u(t): \mathbb{R}^+ \rightarrow \mathbb{R}^d$ ; and the mapping  $f: \mathbb{R}^n \times \mathbb{R}^p \times \mathbb{R}^d \rightarrow \mathbb{R}^n$ . Two servo-control modules, one in the forward path of the control system and the other in the feedback path, are present.

### 9.2.1.1 Rule base

Generally, both the input vector  $u$  and the error vector  $e$  have to be measured (the measurements being denoted by  $u'$  and  $e'$ ) in order to determine a fuzzy tuning inference for the parameter vector  $c$  (the tuning actions being denoted by  $\hat{c}$ ), as for a general parameter-adaptive control algorithm. The rule base of tuning for this general case will contain conditions for both  $e$  and  $u$  and corresponding actions for  $c$ . We restrict our analysis to the case of tuning where the desired performance is specified in terms the error response to a step input. The test input may be applied during a quiescent period of the system and the error response monitored. By adopting this procedure, it is possible to simplify the tuning rule base to include conditions (antecedents) of  $e$  only. Furthermore, it is not necessary to measure  $u$  in this case. While it has these advantages, the use of a step input for establishing the “context” of tuning is clearly a limitation. In particular, in this case, generally, the tuning has to be restricted to quiescent periods of the plant. But the concepts presented here may be extended to other types of tuning inputs and for tuning under non-quiescent plant conditions, provided a corresponding knowledge base of tuning is available. These problems do not arise in applications where the control input is a pulse sequence. Also, the observed attributes may not necessarily be the error state  $e'$  itself, but instead some vector  $\hat{e}$  that is related to  $e'$ . This approach is quite akin to manual tuning, say by the Ziegler–Nicholas approach, and therefore further justified in view of the fact that the rule base that is used in fuzzy logic tuning is assumed to originate and evolve through human experience. In contrast, parameter identification based on input–output data is done not through human experience of tuning but rather using sophisticated analytical relations and algorithms that cannot be realistically extended in general to a linguistic knowledge base for use in fuzzy tuning. Nevertheless, such an extension should not be completely ruled out. For instance, applications have been developed using model-referenced fuzzy adaptive controllers.

Consider a tuning rule base that is given by the general form

$$\text{Else } [ \text{If } E_1^i \text{ Then If } E_2^j \text{ Then If } \dots E_n^k \text{ Then } C_1^a \text{ And } \dots C_p^c ] \quad (9.2)$$

Here,  $E$  and  $C$  denote the “fuzzy” variables of error condition and tuning action, respectively. The attached subscript is the index that identifies a particular variable in a vector of variables. A fuzzy variable can assume several states (e.g., very small, small, medium, large, and very large). The superscript attached to a fuzzy variable is the index that identifies a particular state of the variable. The lower case letters  $e$  and  $c$  denote “crisp” quantities (e.g., actual

measurements or actual tuning values) associated with  $E$  and  $C$ . Suppose that there are  $n$  error-attribute variables  $e_1, e_2, \dots, e_n$  and  $p$  tuning-action variables  $c_1, c_2, \dots, c_p$ . Note that in general, each error attribute (antecedent)  $e_s$  may assume  $m$  discrete fuzzy states  $E_s^1, E_s^2, \dots, E_s^m$ , for  $s = 1, 2, \dots, n$  and similarly, each tuning action (consequent)  $c_q$  may assume  $r$  discrete fuzzy states  $C_q^1, C_q^2, \dots, C_q^r$ , for  $q = 1, 2, \dots, p$ . The integer value  $m$  represents the fuzzy resolution of  $E_s$  and similarly  $r$  represents the fuzzy resolution of  $C_q$ . The tuning rules themselves are assumed to be available in a linguistic form, a typical tuning rule being, “*If the response is moderately oscillatory then slightly decrease the proportional gain and slightly increase the derivative time constant.*” The fuzzy variables  $E_s$  and  $C_q$  are completely represented by their membership functions:  $\mu_{E_s}(e_s): \mathfrak{R} \rightarrow [0, 1]$  and  $\mu_{C_q}(c_q): \mathfrak{R} \rightarrow [0, 1]$ .

It is possible to represent the rule base (9.2) by a fuzzy relation  $R$  whose membership function is  $\mu_R(e, c)$ . To use the *sup-min* composition in the overall rule base (9.2) we first define a useful notation. The set of error attributes  $\{E_1, E_2, \dots, E_m\}$  is denoted by  $E$ . These attributes may respectively assume the fuzzy states  $i, j, \dots, k$  and these states are collectively denoted by the state  $i$ . The fuzzy set of  $E_s$  has a support set and the variable over this support set is denoted by  $e_s$ . Then the set of membership functions of  $E$  which assume the fuzzy state  $i$  is given by

$$\mu_{E^i}(e) \Delta [\mu_{E_1^i}(e_1), \mu_{E_2^i}(e_2), \dots, \mu_{E_n^i}(e_n)] \quad (9.3)$$

For each antecedent (condition) state  $i$  of the set of error attributes  $E$ , according to the rule base (9.2), there is a corresponding consequent (action) state  $j$  for the tuning parameters  $C \Delta \{C_1, C_2, \dots, C_p\}$ . Using the same definition as in (9.3), the associated set of membership functions is denoted by  $\mu_{C^j}(c)$ .

Note that for a particular condition state  $i$  of the error attributes there will be a unique action state  $j$ , assuming, of course, that there are no conflicting rules. But some of the indices in the index set  $j$  may represent a “no change” state because not all the tuning parameters are involved in a given rule. The associated membership values are taken to be unity in this generalized notation. Extending this further, all attributes may not be associated in a particular rule. Hence the set  $i$  may include error attributes that do not exist in a particular rule, and as before, the corresponding membership values are taken to be unity. With this notation, the membership function of the rule base (9.2) is given by

$$\mu_R(e, c) = \sup_{i,j} \min [\mu_{E^i}(e), \mu_{C^j}(c)] \quad (9.4)$$

If  $E$  and  $C$  are used not only as labels of fuzzy variables but also to denote the support sets of the corresponding variables, it should be clear that the rule base  $\mu_R$  is defined in the Cartesian product space  $E \times C \Delta E_1 \times \dots \times E_n \times C_1 \times C_2 \times \dots \times C_p$  as indicated in Figure 9.2.

The first stage of rule dissociation is introduced now. Specifically, if a rule proposes actions on more than one tuning parameter, it is quite valid to consider the action parameters sequentially. This does not imply that the

coupling (or interaction) among the condition and action variables is neglected, but rather, it is considered in a sequential manner. The dissociation of the rules has a secondary benefit as well. Specifically, if the actions can be prioritized according to the order of importance, it will be desirable to arrange the tuning sequence in the same order. For example, in a proportional-integral-derivative (PID) controller, it is appropriate to arrange the tuning actions in the order D-P-I which is the descending order of the natural speed of reaction of these three control actions. In this manner, the rule base (9.2) can be partitioned in such a way that each partition is uniquely associated with one tuning parameter. Then there is a one-to-one correspondence between the rule base partitions and the tuning parameters. For instance, the rule base partition that corresponds to the  $q$ -th tuning parameter has the membership function

$$\mu_{R_q}(e, c_q) = \sup_i \min[\mu_E(e), \mu_{C_q^b}(c_q)] \quad \text{for } q = 1, 2, \dots, p \quad (9.5)$$

which should be compared with equation (9.4). Note that the index  $b$ , which denotes the fuzzy state of  $C_q$  in equation (9.5), as for  $j$  in (9.4), is completely determined by the state  $i$  of the error attributes and as a result  $b$  does not enter as an independent index.

### 9.2.1.2 Compositional rule of inference

As shown in Figure 9.1, suppose that a set of attribute observations is made related to the system behavior. These are crisp quantities in general and, for the purpose of the subsequent fuzzy analysis, they have to be fuzzified. If crisp quantities are represented by “fuzzy singletons” with unity membership grade at the crisp value and zero grade elsewhere in the support set, no information will be lost in fuzzification of a crisp quantity. The resulting fuzzy observations  $\hat{E}\Delta\{\hat{E}_1, \hat{E}_2, \dots, \hat{E}_n\}$  have to be matched with the rule base  $R$  in order to arrive at a fuzzy tuning inference  $\hat{C}\Delta\{\hat{C}_1, \hat{C}_2, \dots, \hat{C}_n\}$ . This is accomplished by applying the compositional rule of inference. Specifically, as in equations (9.4) and (9.5), the sup-min composition is used. It may be easily verified then that the membership function of the fuzzy inference for the  $q$ -th parameter is given by

$$\mu_{C_q^s}(c_q) = \sup_e \min[\mu_E(e), \mu_{R_q}(e, c_q)] \quad \text{for } q = 1, 2, \dots, p \quad (9.6)$$

Since the tuning actions themselves have to be crisp, these fuzzy tuning inferences have to be “defuzzified” using one of the available methods. The *center of gravity method*, or *centroid method*, is widely used, and is given by

$$\hat{c}_q = \frac{\int c \mu_{C_q^s}(c) dc}{\int \mu_{C_q^s}(c) dc} \quad \text{for } q = 1, 2, \dots, p \quad (9.7)$$

in which the integration is carried out over the support set of  $C$ . Tuning of the servo controllers (Figure 9.1) is done using the tuning actions  $\hat{c}\Delta\{\hat{c}_1, \hat{c}_2, \dots, \hat{c}_q\}$ . Obviously some information is lost through defuzzification, but this is an essential practical reality. Furthermore the particular defuzzification procedure as given by equation (9.7) generally tends to make the knowledge-based tuner more robust. While a mathematical proof cannot be given for this statement, the robustness of the center of gravity method has been experienced in practical applications of fuzzy control. Intuitively, since the integration associated with the computation of the center of gravity is an averaging operation, any localized errors or disturbance tends to filter out in the process and will contribute towards the robustness of the tuner.

### 9.2.2 Tuning procedure

The rule base (9.2), in its state-space form, has the disadvantage that it is expressed in terms of the error states as the condition variables. This will inevitably make the rules coupled and rather complex. Furthermore, since knowledge-based tuning relies heavily on human experience and the expertise of the control engineer, it is not advantageous to limit the observed attributes to error state variables alone. Often, other attributes such as percentage overshoot, steady-state error, delay time, and rise time serve as convenient antecedents in the experience of manual tuning of a servo controller, and this is particularly true when the tuning attributes are observed in response to a test input. The use of such familiar performance attributes will not only make the knowledge base used in tuning more realistic and accurate, but also will uncouple (again, through human expertise) the rules in the knowledge base, which can significantly improve the computational efficiency. The concept of rule dissociation and the introduction of resolution relations to link the membership functions of the condition variables with those of the action variables form the basis of the tuning procedure presented here.

#### 9.2.2.1 Rule dissociation

In manual tuning of a control system, a human expert tacitly uses a set of linguistic rules. Knowledge-based fuzzy tuning has its roots in a knowledge base of such rules. In a typical situation, the human expert will observe a particular condition and take one or more tuning actions before turning his attention to another condition. In this sense, the conditions are considered sequentially and repetitively, forming “perception-action cycles.” Many actions may be involved, but one action is considered at a time. For example, in tuning a proportional-integral-derivative (PID) servo, if the expert notices an oscillatory (instability) condition he would first increase the rate feedback parameter and decrease the loop gain and subsequently might take appropriate tuning actions for a speed of response problem. This type of sequential observation of condition variables will lead directly to a dissociation of rules. Furthermore, in practice, the observation–action cycles are incremental in nature. For example, if a strong instability were observed, a large tuning

adjustment would be made, and subsequently, should the condition become moderate, the tuning actions would be toned down. Under these conditions, the linguistic tuning relation between a particular attribute ( $E_s$ ) and the corresponding action ( $C_q$ ) may be viewed as a sequence of monotonically incrementing rules; specifically,

$$R_{s,q} : \underset{i=0}{\overset{m-1}{\text{Else}}} [\text{If } E(i) \text{ Then } C(i)] \quad (9.8)$$

The subscripts  $s$  and  $q$  have been dropped in the rules given by (9.8), for convenience. The membership function of  $R_{s,q}$  is formed by the sup-min composition:

$$\mu_{R_{s,q}}(e, c) = \sup_{i=0}^{m-1} \min [\mu_{E(i)}(e), \mu_{C(i)}(c)] \quad (9.9)$$

The overall rule base is given by

$$R = \bigcup_{s,q} R_{s,q} \quad (9.10)$$

The analytical basis of decoupling a set of multi-antecedent rules into single-antecedent rules has been addressed in the chapter on fuzzy control. This further supports the rule dissociation implied by (9.8).

### 9.2.2.2 Resolution relations

The monotonically incrementing characteristic that is exhibited in the rule base (9.8) will lead to a quantification of fuzzy resolution. In connection with the rule base (9.2), it was mentioned previously that the total number of possible fuzzy states for a fuzzy variable will represent its fuzzy resolution. For example,  $E_s$  and  $C_q$  in (9.2) can assume respectively  $m$  and  $r$  fuzzy states. Unfortunately, in (9.2) the resolutions are fixed at the outset, in the problem definition, and the rule base must be modified and the entire problem reformulated if one wishes to change the resolution levels. The incremental rule base given by (9.8) is attractive in this respect, because here the fuzzy resolution is linked to the index  $i$  and can be refined arbitrarily and varied during operation of the tuner, without having to reformulate the problem. The concept of resolution relations is introduced now to formalize fuzzy resolution and the tuning procedure.

Suppose that the membership function  $\mu_{E(i)}(e)$  has a modal value  $\bar{e}_i$  and similarly  $\mu_{C(i)}(c)$  has a modal value  $\bar{c}_i$ . These membership functions are assumed to be *unimodal*, which is justified because one fuzzy state ( $i$ ) is considered at a time, and each state is represented by a specific mode. Typically the modal value is the value of the variable at the peak membership grade, and this usage is particularly convenient when triangular membership functions are used. An alternative modal value (e.g., the centroid of the membership function) may be used if its use can be justified; for example, in the case of a trapezoidal membership function. This may lead to increased computational complexity but the concepts presented here will be still applicable.

A general *resolution relation* (or *resolution curve*) for  $E(i)$  and  $C(i)$  may be expressed by the *resolution functions*:

$$\bar{e}_i = r_e(i) \quad i = 0, 1, \dots, m - 1 \quad (9.11)$$

$$\bar{c}_i = r_c(i) \quad i = 0, 1, \dots, m - 1 \quad (9.12)$$

in which  $r_e$  and  $r_c$  are monotonic functions in the index  $i$ , and these functions will determine the fuzzy resolution of the corresponding variables. Specifically, one could emphasize various tuning levels by appropriately selecting the resolution functions. For example, if linear functions are used, all tuning actions are uniformly emphasized and if exponential functions are used, large changes are more heavily emphasized than small changes. Polynomial resolution functions of the form

$$\bar{e}_i = a_e i^{m_e} \quad i = 0, 1, \dots, m - 1 \quad (9.13)$$

$$\bar{c}_i = a_c (i^{m_c} - \beta) \quad i = 0, 1, \dots, m - 1 \quad (9.14)$$

are normally appropriate. Note that the coefficients  $a_e$  and  $a_c$  are positive parameters representing intensities and the exponents  $m_e$  and  $m_c$  are positive (typically integer) parameters representing the degree of nonlinearity. The parameter  $\beta$  provides an offset which will provide a means of reducing the magnitudes of the tuning parameters when appropriate, under quiescent conditions. Care has to be exercised in selecting the value of this parameter, because with large values unsteady behavior may result.

A resolution relation is the relationship between the resolution function of a condition variable and the resolution function of an action variable. By means of a resolution relation, fuzzy resolution can be conveniently modified on line during operation of the knowledge-based tuner without having to reformulate the entire problem. In practice, lower and upper bounds have to be introduced to the sequences in equations (9.13) and (9.14) in order to incorporate physical constraints or stability bounds of the tuning parameters. These bounds may be known through experience.

Resolution relations can be introduced even if complete rule dissociation is not carried out. In this case the resolution relations are multivariable and will relate the modal values of several attribute variables to the modal value of an action variable; for example,

$$\bar{c}_{i,h,\dots,k} = f_r(\bar{e}_i, \bar{e}_j, \dots, \bar{e}_k) \quad (9.15)$$

which is termed a *resolution surface*.

To summarize, a fuzzy state is represented by an indexed and unimodal membership function, and the membership function has a modal value, which is also indexed. A *resolution function* provides an analytical representation of the fuzzy states (and fuzzy resolution) of a condition variable or an action variable, which are related by a fuzzy rule. Specifically, a resolution function is a function of the indexing variable, for a modal value. It gives the

progression of the modal values, which characterize the associated unimodal membership functions. In this latter sense it affords some form of “analytical interpolation” between alternative fuzzy states in an appropriately partitioned rule base. An analytical relationship between the indexed modal values of a condition variable and a corresponding action variable (i.e., the relationship between the condition resolution function and the action resolution function) provides a *resolution relation* (or *resolution curve*). Resolution relations that are nonlinear and complex may be conveniently expressed in terms of appropriate resolution functions.

### 9.2.2.3 Tuning inference

Suppose that the membership function of the dissociated rule base is determined as given by equation (9.9). The compositional rule of inference has to be applied to this rule base, with a crisp measurement  $\hat{e}_s$  of the corresponding attribute  $E_s$ . The measurement may be represented as a *fuzzy singleton*  $s(e - \hat{e}_s)$ :

$$\begin{aligned}\mu_{\hat{E}_s}(e)\Delta s(e - \hat{e}_s) &= 1 \quad \text{for } e = \hat{e}_s \\ &= 0 \quad \text{elsewhere}\end{aligned}\tag{9.16}$$

The compositional rule of inference, as given by equation (9.6), can be applied here; thus

$$\begin{aligned}\mu_{C_q}(c) &= \sup_e \min[\mu_{\hat{E}_s}(e), \mu_{R_{s,q}}(e, c)] \\ &= \sup_e \min[s(e - \hat{e}_s), \mu_{R_{s,q}}(e, c)]\end{aligned}$$

which, in view of equation (9.16) becomes

$$\mu_{C_q}(c) = \mu_{R_{s,q}}(\hat{e}_s, c)\tag{9.17}$$

Finally a crisp tuning action  $\hat{c}_q$  may be computed according to equation (9.7). Note that the tuning actions are incremental and not continuous, and an increment is governed by the resolution relation that is employed, as discussed.

### 9.2.2.4 Accuracy versus fuzzy resolution

Appropriate use of the level of fuzzy resolution is important in a knowledge-based fuzzy control system. Generally speaking, the higher the fuzzy resolution that the variables can assume, the more accurate the control a system can achieve. However, as the resolution increases, generally, the computational time will also increase. This may affect the performance of the control system, for example, with regard to the control bandwidth, or require more expensive hardware. More importantly, the stability of the system may be affected by the increased fuzzy resolution. For example, if a low fuzzy resolution is used in the system, the number of rules that define different input conditions

would be limited, and there is a good possibility that no rules will define certain input conditions or states. Such undefined or ambiguous situations can lower the efficiency of the fuzzy controller. This problem can be avoided or partially resolved by increasing the fuzzy resolution, and consequently the accuracy of the system will improve, but the operating bandwidth will decrease due to increased computational effort. On the other hand, only a coarse tuning action may be generated by a system with low fuzzy resolution. Hence, it is possible that the system will exhibit a hunting response about the desired operating point.

To illustrate this point further, assume that a series of “well-behaved” input–output data is available, which could be generated, say, by a human or well-tuned automatic controller. Fuzzy associative memory (FAM) rules can be obtained using the adaptive vector quantization (AVQ) algorithm with differential competitive learning (DCL). In control applications, given the crisp values of both input and output variables, the DCL-AVQ method can be used to adaptively cluster these data vectors to estimate the underlying control surface (continuous rules) which generated them. The product space is then partitioned into FAM cells, each one representing a single FAM rule. The maximum number of FAM cells is determined by the product  $mp$  as defined in equation (9.2) and the number of FAM cells will determine the smoothness of the control surface. A smoother control surface implies that, given the data for input conditions, finer tuning actions could be generated, or, in other words, a more accurate relationship between the performance conditions and the tuning actions could be established.

On the other hand, as mentioned earlier, the processing speed is directly affected by the fuzzy resolution, and accordingly, the heavy computational burden resulting from a high fuzzy resolution can slow down the response, thereby reducing the operating bandwidth. In order to analyze the effect of fuzzy resolution on the accuracy, however, suppose that the computational efficiency is not affected by the fuzzy resolution. In a practical implementation, this may be accomplished by first developing a crisp decision table off line, and then using this table in a look-up mode to determine the crisp control actions during operation. However, with a decision table, the number of discretization levels would be limited by the memory capacity of the controller – only a coarse control surface could be achieved with a few discretization levels.

### 9.2.3 Illustrative example

To illustrate the application of the concepts presented here, consider the example of a nonminimum phase plant with transport delay, as shown in Figure 9.3. The plant is controlled directly by a proportional-integral-derivative (PID) loop. Knowledge-based tuning of the PID parameters: the proportional gain  $K_p$  and integral gain  $K_i$ , and the derivative time constant  $\tau_d$ , is studied. The plant parameters used in this example are  $K = 1.0$ ,  $\tau_1 = 2.0$ ,  $\tau_2 = 0.5$ ,  $\tau_3 = 0.1$ , and  $\tau = 0.2$ . Also, the controller parameters  $K_f$  and  $\tau_f$  are fixed at 1.0 and 0.02, respectively.

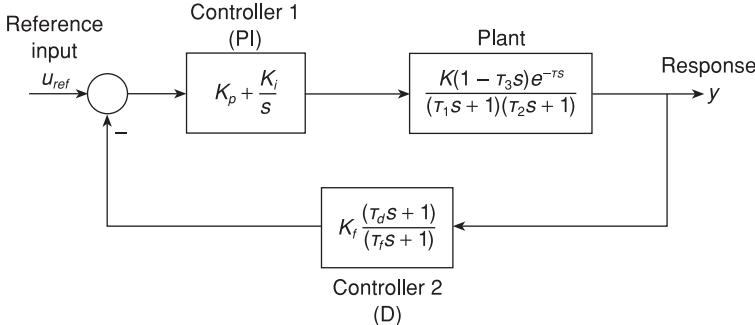


Figure 9.3: A nonminimum phase plant with transport delay and a PID servo

#### 9.2.3.1 Resolution relation

The attributes of the system response that are used in tuning the PID parameters are steady divergence, overshoot, oscillation amplitude, speed of response, and steady-state error, for a step input command. The first three attributes are primarily stability indicators and the fourth one is a system bandwidth (speed or maximum frequency of interest of the response) indicator. In the present example, overshoot and oscillation amplitude are treated as the same attribute and only the latter is included in the tuning scheme. Furthermore, rise time (time to reach the desired level for the first time) is considered as the speed of response attribute. The tuning criteria used are presented in Table 9.1. In a conventional scheme, fuzzy terms such as small, moderate, large, and very large have to be introduced in these criteria. In the present scheme, however, this fuzzy resolution is implicitly incorporated into the resolution relation.

The membership functions of both condition and action variables are given by the triangular function

$$\begin{aligned}\mu_{X(i)}(x) &= \max\left(0, \frac{x - \bar{x}_{i-1}}{\bar{x}_i - \bar{x}_{i-1}}\right) \quad \text{for } x \leq \bar{x}_i \\ &= \max\left(0, \frac{\bar{x}_{i+1} - x}{\bar{x}_{i+1} - \bar{x}_i}\right) \quad \text{for } x \geq \bar{x}_i\end{aligned}\tag{9.18}$$

Table 9.1: The tuning criteria for the PID servo

Performance attribute	Tuning action for		
	$T_d$	$K_p$	$K_i$
Steady divergence	Increase	Decrease	Decrease
Overshoot/oscillations	Increase	Decrease	Decrease
Speed of response	No change	Increase	Decrease
Steady-state error	No change	Decrease	Increase

where  $X$  stands for either  $E$  or  $C$  and  $\bar{x}_i$  denotes the modal value. Note that  $E$  denotes any one of the four performance attributes that are listed in the first column of Table 9.1, and  $C$  denotes any one of the three tuning parameters ( $\tau_d$ ,  $K_p$ , and  $K_i$ ) that are listed in the top row of Table 9.1. As discussed before, only one performance attribute and one tuning parameter are considered at a given instant of tuning and all applicable such pairs are considered sequentially. In each tuning cycle, the order of the sequence is arranged according to some desirable criterion of priority. In the present example, tuning actions for a given performance attribute are carried out in the order  $\tau_d \rightarrow K_p \rightarrow K_i$  as dictated by the reaction time of the corresponding control actions (derivative action being faster than the integral action). Furthermore, in the present example, the order in which the performance attributes are considered during tuning is arranged in the order given in Table 9.1 (i.e., Divergence  $\rightarrow$  Oscillations  $\rightarrow$  Speed  $\rightarrow$  Steady-State Error). This order is further justified according to practical criteria; for example, undesirable instabilities have to be corrected before adjusting the speed of response or compensating for any steady-state error. It should be mentioned here that every performance attribute might not be active in all tuning cycles. For example, the two tuning actions for  $K_p$  and  $K_i$  corresponding to the steady-state error attribute will not be active unless the steady-state is reached, and then, too, unless the specified performance requirement on the steady-state error is violated.

For simplicity, linear resolution functions of the index  $i$  are chosen (i.e.,  $m_e = 1 = m_c$  in equations (9.13) and (9.14)). Now, in order to establish a membership function for each tuning criterion, as given by equation (9.9), first we form  $\mu_{R_i}(e, c) = \min[\mu_{E(i)}(e), \mu_{C(i)}(c)]$  which becomes

$$\begin{aligned} \mu_{R_i}(e, c) &= \max\left(0, \min\left[\frac{e - \bar{e}_{i-1}}{\bar{e}_i - \bar{e}_{i-1}}, \frac{c - \bar{c}_{i-1}}{\bar{c}_i - \bar{c}_{i-1}}\right]\right) \quad \text{for } \begin{array}{l} e \leq \bar{e}_i \\ c \leq \bar{c}_i \end{array} \\ &= \max\left(0, \min\left[\frac{e - \bar{e}_{i-1}}{\bar{e}_i - \bar{e}_{i-1}}, \frac{\bar{c}_{i+1} - c}{\bar{c}_{i+1} - \bar{c}_i}\right]\right) \quad \text{for } \begin{array}{l} e \leq \bar{e}_i \\ c > \bar{c}_i \end{array} \\ &= \max\left(0, \min\left[\frac{\bar{e}_{i+1} - e}{\bar{e}_{i+1} - \bar{e}_i}, \frac{c - \bar{c}_{i-1}}{\bar{c}_i - \bar{c}_{i-1}}\right]\right) \quad \text{for } \begin{array}{l} e > \bar{e}_i \\ c \leq \bar{c}_i \end{array} \\ &= \max\left(0, \min\left[\frac{\bar{e}_{i+1} - e}{\bar{e}_{i+1} - \bar{e}_i}, \frac{\bar{c}_{i+1} - c}{\bar{c}_{i+1} - \bar{c}_i}\right]\right) \quad \text{for } \begin{array}{l} e > \bar{e}_i \\ c > \bar{c}_i \end{array} \end{aligned} \tag{9.19}$$

Subsequently, the *supremum* operation is carried out over the entire range of the index  $i$  which results in a membership function of the form shown in Figure 9.4(a), with a support set (footprint) of the form shown in Figure 9.4(b). This function is programmed into the tuning scheme. Since the resolution functions are linear in  $i$ , with zero offset, the modal values  $\bar{e}_i$  and  $\bar{c}_i$  can be expressed as  $e_i = i\bar{e}$  and  $c_i = i\bar{c}$ , where  $\bar{e}$  and  $\bar{c}$  are appropriate scaling parameters for the corresponding condition and action variables. Consequently, just one generic membership function need be programmed as a subroutine in a nondimensional form for all combinations of tuning criteria. Finally,

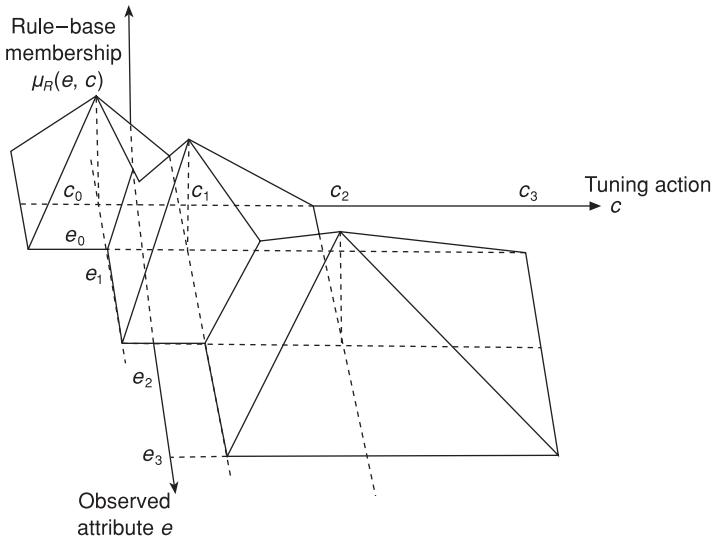


Figure 9.4 (a): Membership function of a tuning criterion (rule)

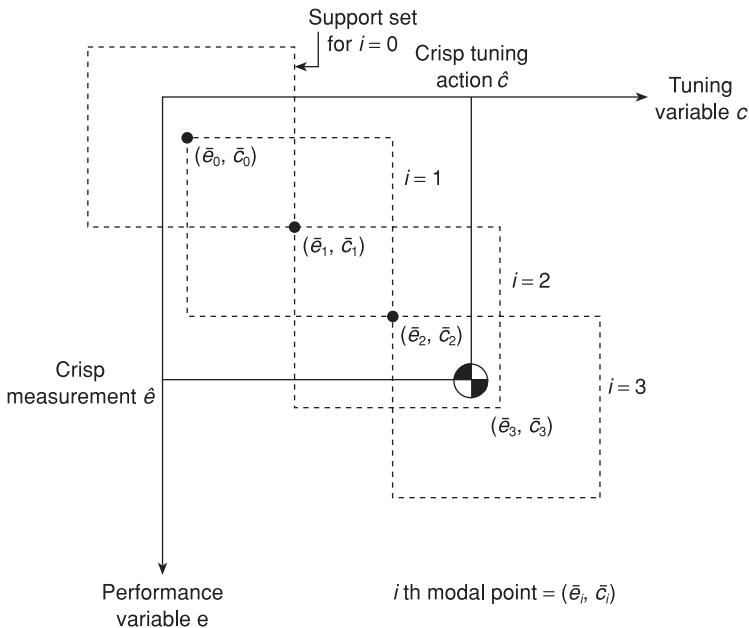


Figure 9.4 (b): The support set of a tuning rule

incremental tuning actions are computed using equation (9.7), through simple trapezoidal integration.

To further illustrate the main steps of the tuning program, consider the condition-action pair: (Divergence,  $\tau_d$ ). First, the scaling parameters  $\bar{e}_i$  and  $\bar{c}_i$  for this pair have to be chosen appropriately. This is usually done through

past experience. For example, a performance specification  $\bar{e}_{max}$  is specified for steady divergence. Then, when the error exceeds  $\bar{e}_{max}$ , if only a mild tuning action is desired, we may choose  $\bar{e}$  greater than or equal to  $\bar{e}_{max}$ . If a stronger action is desired,  $\bar{e}$  may be chosen smaller than  $\bar{e}_{max}$ . Subsequently, the values of  $\bar{e}_{max}$  and  $\bar{e}$  may be adjusted in an appropriate manner through past experience (by simulation or experimentation). In the present example we have used  $\bar{e}_{max} = 1.0$  and  $e = 0.5$  for “Divergence.” Similarly, for  $\tau_d$  a starting value  $\tau_{d_0}$  and a modal scaling value  $\bar{c}$  have to be chosen. For example,  $\tau_{d_0}$  may be chosen as the lowest value that will not damage the system. Since  $\bar{c}$  scales the tuning increment, it should depend on both the required magnitude of tuning and the frequency at which tuning is done (tuning bandwidth). In the present example we have used  $\tau_{d_0} = 0.0$  and  $\bar{c} = 0.005$  for the derivative action in the context of steady divergence. Once these numerical values are decided, the program steps for tuning, in relation to the present condition-action pair, are as follows:

- (1) Compute (measure) the error response  $e$ .
- (2) If  $e/\bar{e}_{max} < 1.0$  Then No Action; and proceed to check the next performance attribute (i.e., oscillations)  
If  $e/\bar{e}_{max} \geq 1.0$  Then Go to Step 3.
- (3) Compute  $e/\bar{e}$ .
- (4) Call the subroutine for nondimensional membership function  $\mu_R(e, c)$  of the rule base, and obtain the nondimensional action-value function corresponding to the nondimensional  $e$  computed in Step 3 (See Figure 9.4(b)).
- (5) By numerical integration (according to equation (9.7)) compute the centroid of the action value function obtained in Step 4 (See Figure 9.4(b)).
- (6) Scale the  $c$  value computed in Step 5 by  $\bar{c}$ .
- (7) Increment  $\tau_d$  by the value computed in Step 6. If the hard limits of  $\tau_d$  are exceeded,  $\tau_d$  is set back to the old value.
- (8) Proceed to tune the next parameter (i.e.,  $K_p$ ).

Note that all computations are done on discrete (not continuous) data. Specifically in the present example, 100 data points have been used for each membership function corresponding to each index value. Also, the response integration and control signal computation are done in period of 0.02 s. This corresponds to a “control bandwidth” of 50.0 Hz.

### 9.2.3.2 Stability region

For a nominal system (plant and the PID servo loop) given in Figure 9.3 with  $K_p = 5.0$ ,  $K_i = 2.0$ , and  $\tau_d = 0.01$ , the stability regions were determined, assuming the first order Prade' approximation for the transport delay. The stability regions were computed to be  $K_p : [0.445, 8.003]$ ,  $K_i : [0, 4.696]$  and  $\tau_d : [0, 1.089]$ . The eigenvalues of the nominal system are  $-0.2712 \pm j 2.0402$ ,  $-0.3875$ ,  $-12.3018$ , and  $-49.5182$ . It follows that the nominal system is

stable. Note that the first three poles are contributed by the two time constants of the plant and by the integral control action. The fourth pole results primarily from the Prade' approximation to the transport delay and the fifth pole is due mainly to the time constant  $\tau_f$  in Controller 2. The tuner is expected to push the system into the stability region computed here, even when the tuning process is started outside the region (unstable system). This is illustrated in the next section.

#### 9.2.3.3 Tuning results

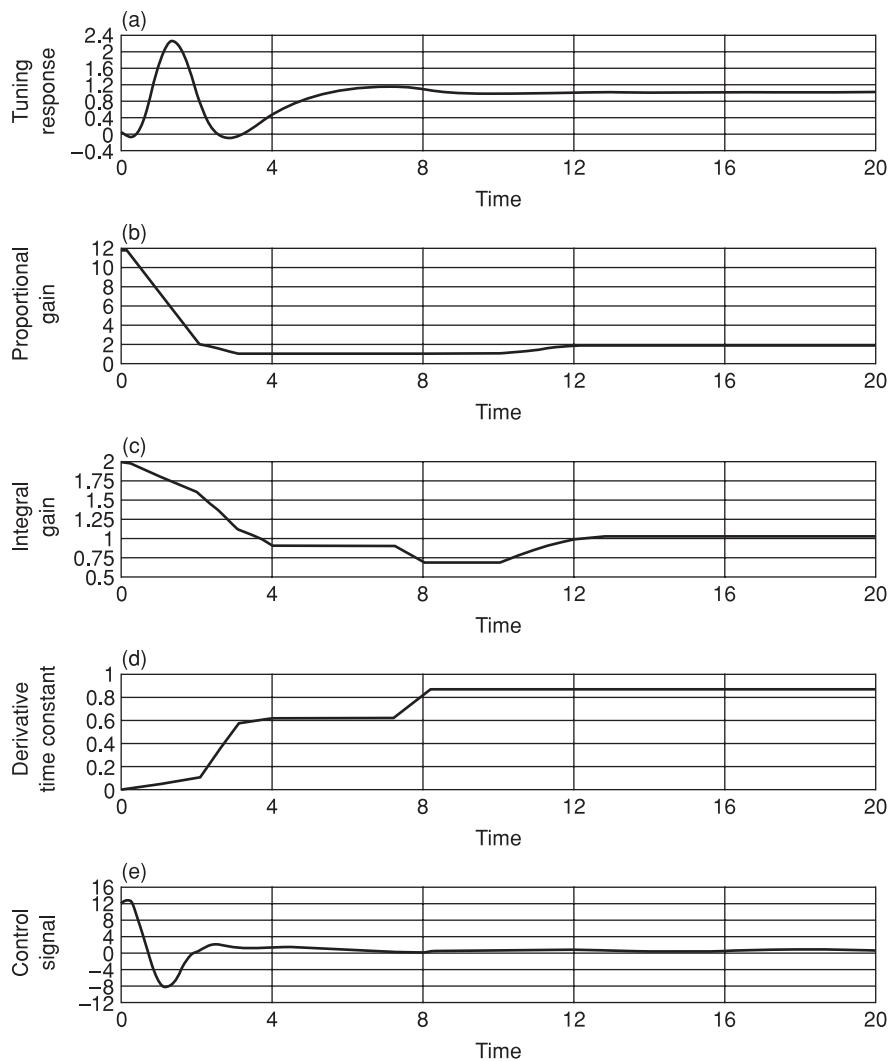
The tuner was activated with a unit step input to the system and with the initial PID parameters,  $K_p = 12.0$ ,  $K_i = 2.0$  and  $\tau_d = 0.01$ . Note that, according to the stability regions, the initial system is unstable. The control bandwidth was 50 Hz and the tuning bandwidth used was 1 Hz. This means that the control signal is updated at every 0.02 s and each tuning decision is distributed over a period of 1 s. Note that, even though the performance attributes are checked at the rate of the tuning bandwidth (every 1.0 s), this does not mean that tuning actions are made for each one of these attributes at this rate. For example, the actions corresponding to the steady-state error are activated if and only if the following two conditions are satisfied: (9.1) The steady-state is reached. (9.2) The steady-state error exceeds the specified tolerance value. The total duration of tuning will depend on many factors such as the plant characteristics, controller characteristics, the magnitudes of various tuning actions, and the frequency of the tuning actions. The tuning simulation was carried out for a duration of 20 s.

The response of the system with the tuner active is shown in Figure 9.5. Observe that the tuned condition is reached in less than 14 seconds in this case. It is clear that the tuner detects the stability quite early in the response and takes corrective measures and subsequently goes on to making finer adjustments including one for steady-state error (towards the middle and end of the simulation when the steady-state is reached). The control signal that goes into the plant is shown in Figure 9.5(e).

Figures 9.6(a) and (b) show the response of the system to the pulse sequence of Figure 9.6(c). When the PID parameters were fixed at the mis-tuned initial values, as expected, the response became quite unstable just a fraction of the way into the first pulse, as shown in Figure 9.6(a). But, when tuned PID parameters were employed, the response was found to be quite satisfactory, as given in Figure 9.6(b). The control signal into the plant corresponding to the reference pulse command, for the tuned system, is shown in Figure 9.6(d).

### 9.3 Supervisory control of a fish processing machine

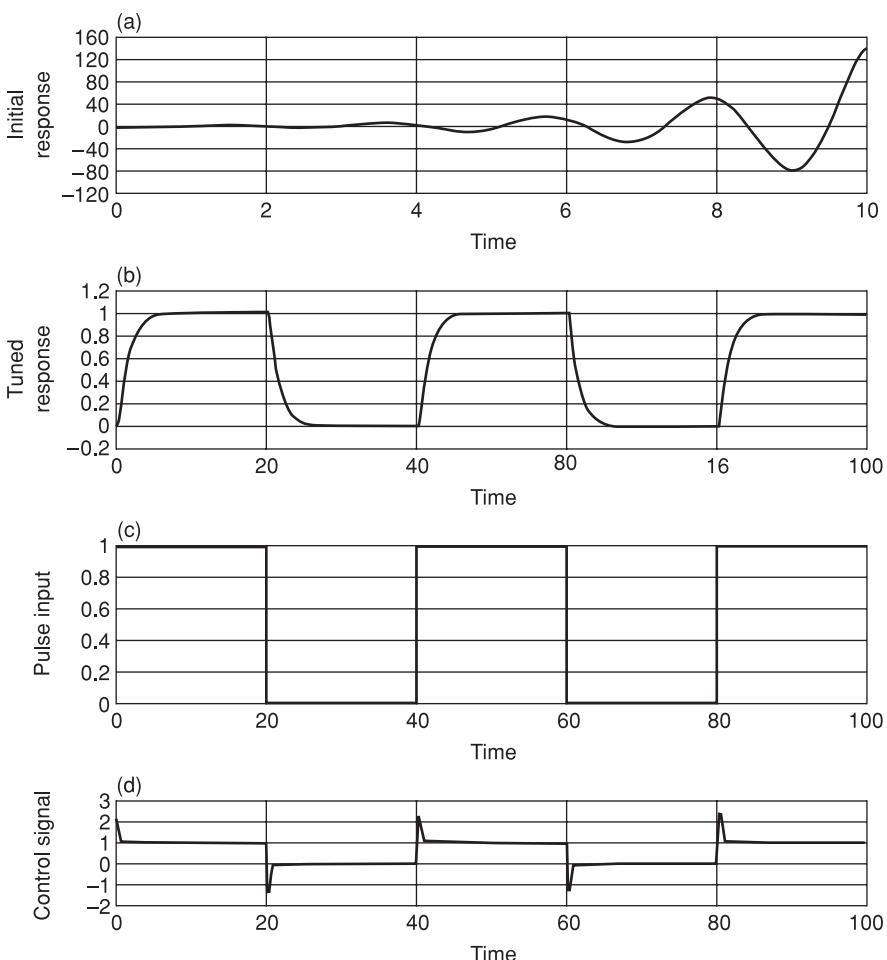
Fish processing, a multi-billion dollar industry in North America alone, by and large employs outdated technology. Wastage of the useful fish meat during processing, which reaches an average of about 5%, is increasingly becoming a matter of concern for reasons such as dwindling fish stocks and



**Figure 9.5:** The tuning response of the system with a step input: (a) System response. (b) Proportional gain. (c) Integral gain. (d) Derivative time constant. (e) Control signal to the plant

increasing production costs. Due to the seasonal nature of the industry, it is not cost effective to maintain a highly skilled labor force to carry out fish processing tasks and to operate and service the processing machinery. Due to the rising cost of fish products and also the diverse tastes and needs of the consumer, the issues of product quality and products-on-demand are gaining prominence. To address these needs and concerns, the technology of fish processing should be upgraded so that the required tasks could be carried out in a more efficient and flexible manner.

### 9.3 Supervisory control of a fish processing machine



**Figure 9.6:** System response to a pulse input: (a) Before tuning. (b) After tuning. (c) Input pulse signal. (d) Control signal

A machine termed the Iron Butcher, which was originally designed at the turn of the twentieth century and has not undergone any major changes, is widely used in the fish processing industry for carrying out the head cutting operation of various species of salmon. Motivated by the potential for improvement of this machine, specifically with regard to waste reduction, productivity and production flexibility, a project was undertaken in the Industrial Automation Laboratory, University of British Columbia, to develop a machine for fish cutting. The machine has been designed, integrated, tested, and refined. The test results have shown satisfactory performance. The machine uses an intelligent supervisory control system that is based on a layered architecture in which sensory and direct control tasks of the machine components are carried out at the bottom layer, and the monitoring, tuning, and quality assessment tasks are accomplished by knowledge-based upper layers

using fuzzy logic. The present section describes the machine and the development of the supervisory control system. Development of the knowledge-based modules of the system, which employ fuzzy logic for knowledge representation and decision-making, is described. Experimental procedures and performance evaluation are presented as well.

### 9.3.1 Machine features

A machine has been developed for fish cutting, which possesses the following important features:

- (1) High cutting accuracy; obtained using mechanical fixtures, positioners, tools and associated sensors, actuators, and controllers which have been properly designed and integrated into the machine.
- (2) Improved product quality; achieved through high-accuracy cutting and also through mechanical designs that do not result in product damage during handling and processing, along with a quality assessment and supervisory control system which monitors the machine performance, determines product quality, and automatically makes corrective adjustments.
- (3) Increased productivity and efficiency; attained through accurate operation and low wastage, with reduced downtime and less need for reprocessing of poorly processed fish.
- (4) Flexible automation; requiring fewer workers for operation and maintenance than the number needed for a conventional machine, which is possible due to the capabilities of self-monitoring, tuning, reorganization, and somewhat autonomous operation, as a result of advanced components, instrumentation, and the intelligent and hierarchical supervisory control system of the machine.

The hardware structure of the machine is schematically shown in Figure 9.7. A brief description of this machine is given now. The fish are placed one by one on the conveyor at the feeding end of the machine. Subsequently, as each fish passes through the primary imaging station, an image of the fish is generated by a CCD camera and captured by a SHARP GPB image processor. Simultaneously, the thickness of the fish in the head region is sensed through an ultrasonic position sensor. The image is processed by the GPB card, which occupies one slot of the primary control computer, a personal computer (PC). The main image processing steps are filtering, enhancing, thresholding, and labeling of the image objects. In this manner, the gill plate and correspondingly, the collarbone of the fish are identified and gauged. This information is then used to determine the desired position of the cutter. As the fish leaves the primary imaging station it enters a positioning platform whose purpose is to deliver each fish symmetrically into the cutter unit. The desired vertical position of the delivery platform is determined according to the thickness

### 9.3 Supervisory control of a fish processing machine

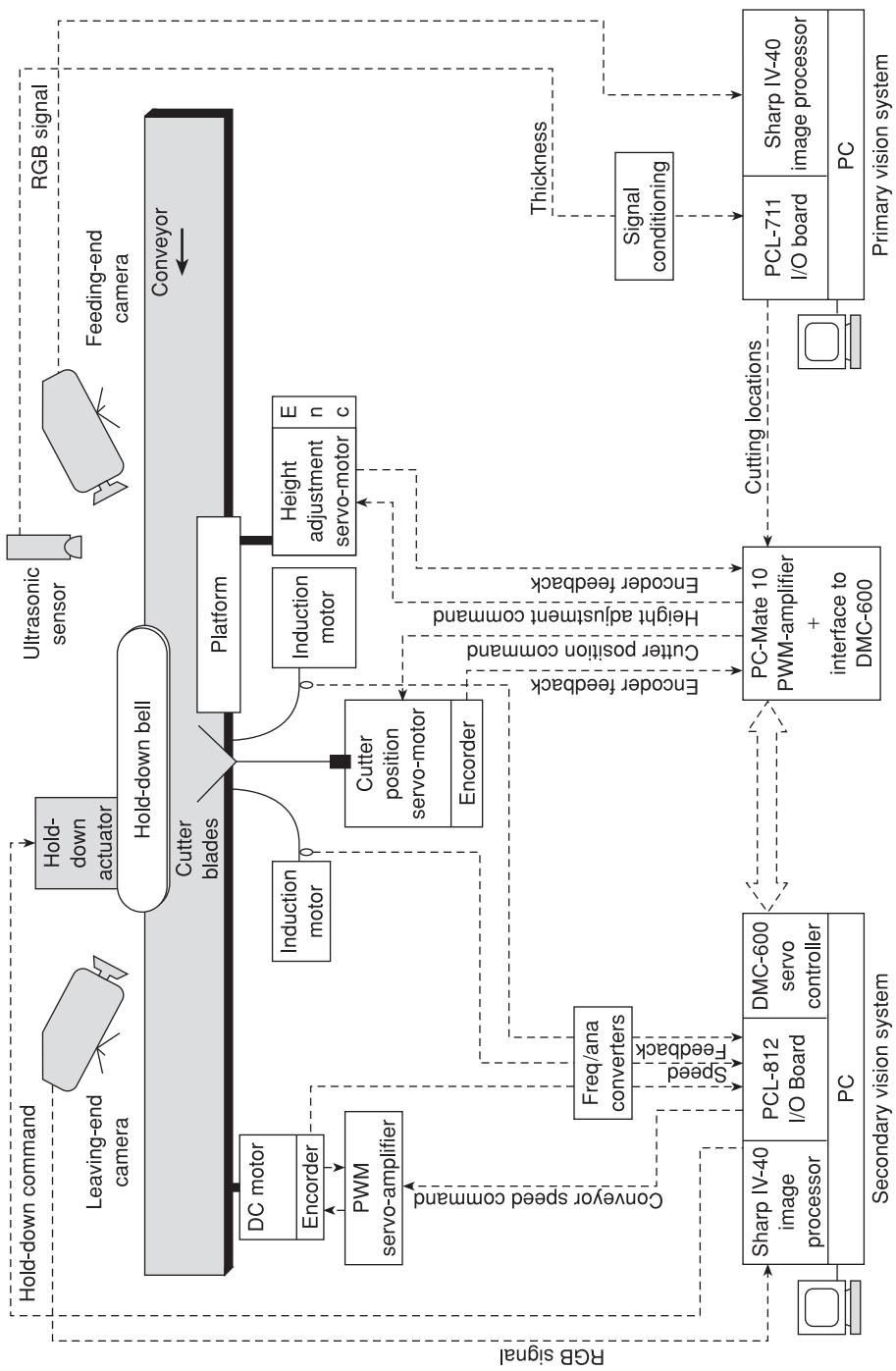


Figure 9.7: The hardware architecture of the machine

of the particular fish, as measured by an ultrasonic sensor and read into the control computer through a PCL-711 Input/Output board. The platform is driven by a DC servomotor through a lead-screw and nut arrangement. The desired position is the reference with which the actual position, as measured by the optical encoder of the servomotor, is compared to generate the servo signal for platform control. The cutter unit consists of a pair of circular blades arranged in a V-configuration and driven by two remotely placed, three-phase induction motors, which are connected to the blades through long flexible shafts.

The V-configuration allows the cutter to reach into the head region below the collarbone and recover the useful meat that is present in this region. In a manner similar to the delivery platform, the cutter is positioned by a DC servomotor through a lead-screw and nut arrangement. The two servomotors, one for the delivery platform and the other for the cutter, are controlled by means of a GALIL DMC-600 two-axis controller which receives reference commands from the control computer and generates input signals for the two pulse-width-modulated (PWM) amplifiers which drive the motors.

There is a hold-down mechanism for fish along the conveyor, starting from the primary imaging station and ending at the cutting station. This device consists of a matrix of spring-loaded rollers that can apply a desired distribution of forces on the fish body so that each fish is properly restrained. The hold-down mechanism will ensure that the configuration of a fish, as measured by the imaging system, does not change as it reaches the cutter and also will provide an adequate holding force during the cutting process itself. Clearly, the restraining forces should not crush or damage the fish. The hold-down force is controlled by the supervisory control system of the machine.

A secondary CCD camera has been mounted on the product-exit side of the cutting station. The images generated by this camera are captured and processed by a dedicated SHARP GPB card that is mounted within a PC. This computer, which hosts the secondary vision system, also serves as the supervisory-control computer. The conveyor is driven by a variable-speed DC motor through a PWM amplifier, according to a speed command given by the supervisory control computer. This computer reads the optical encoders of the conveyor motor and the cutter induction motors, through frequency-to-analog conversion circuitry and a PCL-812 Input/Output board; and from this information the conveyor speed and the loads at the two cutter blades are determined. This computer also receives optical encoder signals from the cutter positioning and platform positioning servomotors. The control signals generated by the supervisory control computer, on the basis of various sensors' information and other knowledge, are also transmitted to the actuators through the same PCL-812 board.

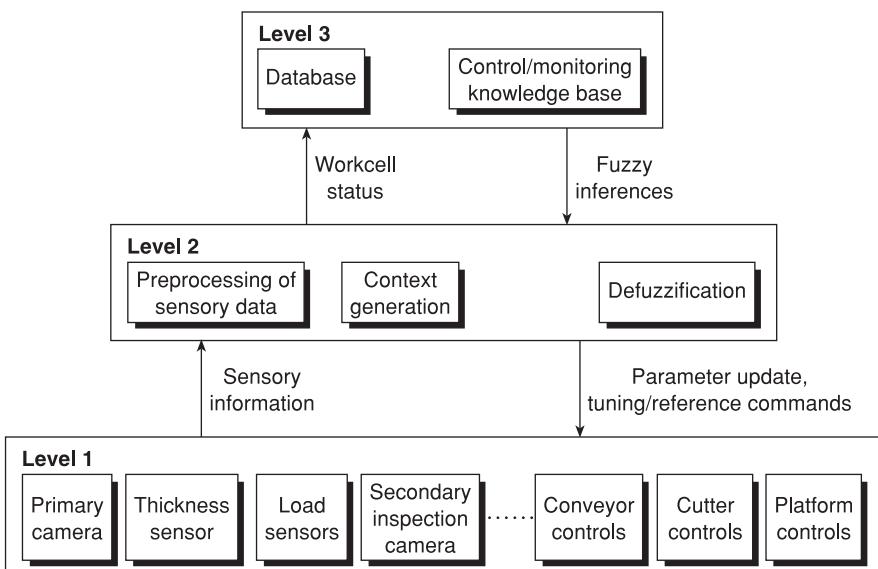
### 9.3.2 Supervisory control system

The overall control system of the fish processing machine has a variety of functions, which may be organized according to an appropriate criterion. The criteria that would be appropriate here include the required speed of

control (control bandwidth), the level of detail of the information that is handled (information resolution), crispness of the knowledge that is involved (fuzzy resolution), and the level of intelligence that is needed to carry out a particular function. It may not be necessary to incorporate all such criteria in a particular system since there exist at least intuitive relationships among these criteria. This is the case particularly if the control system is organized into a hierarchical architecture.

A fully automated fish processing machine will incorporate a multitude of components that have to be controlled and properly coordinated in order to obtain satisfactory overall performance. The plant operation will involve many different types of actions such as sensing, image processing, direct control, performance monitoring, and component coordination. Furthermore, an intelligent agent operating in a supervisory capacity will be able to perform such high-level assignments as goal description, decomposition of a goal into sub-tasks, and sub-task allocation. A multilayered architecture may be developed for the machine by considering the nature and requirements of these various functions. In the present application the three-level architecture that is schematically shown in Figure 9.8 has been implemented. This structure may be considered as a supervisory control system. The lowest layer consists of high-resolution sensors, process actuators, and associated direct controllers. The intermediate layers primarily carry out information preprocessing, determination of the machine status, and command transfer for operating and modifying the performance of the system components. The top layer carries out system monitoring and high-level supervisory control activities.

The bottom layer of the control system of Figure 9.8 carries out direct control of system components in a conventional sense. Specifically, the



**Figure 9.8:** Hierarchical supervisory control system of the machine

component responses are sensed at high resolution, control commands are generated on this basis at high speed, according to hard (i.e., crisp) algorithms, and the components are directly actuated using the control commands. These high-resolution, high-bandwidth, direct control operations do not need a high degree of intelligence for their execution. Specifically, in the prototype machine that has been developed by us, the following sensory information is acquired in the bottom layer:

- (1) Position and speed information of the positioning servo systems and the conveyor, from the respective optical encoders.
- (2) Cutter load information from the induction motors, which drive the cutter blades.
- (3) Visual images from the position sensing CCD camera (primary).
- (4) Visual images from the task monitoring CCD camera (secondary).
- (5) Thickness information from the ultrasound height sensor.

The associated raw signals are characterized by the following properties:

- (1) Large quantities of data points are collected.
- (2) Data points are generated in rapid succession.
- (3) The sensory signals have a high resolution and tend to be quite precise, in the absence of noise.

For example, an image may contain about 250 kilobytes of information, and may be captured every 0.5 seconds. This will represent an information rate of 500 kB/s for this optical sensor alone. Within the bottom layer, some sensory signals would be directly used, without “intelligent” preprocessing, for device control. This is the case with the encoder signals from the servomotors and the ultrasonic thickness sensor, which are directly used in feedback for positioning control.

The top layer of the hierarchical system shown in Figure 9.8 is considered the most intelligent level of the machine. It is responsible for monitoring the overall performance of the system, including product quality, production efficiency, and throughput rate. On this basis, it would make decisions on appropriate adjustments and modifications to the system so as to improve the machine performance and to maintain within specification the operating conditions of various components in the machine. The nature of these objectives is such that they are typically qualitative and descriptive. As such, it is difficult to transcribe them into a set of quantitative specifications. The required actions to achieve these goals are typically based on the knowledge, experience, and expertise of human operators and skilled workers. Consequently, they may be available as a set of linguistic statements or protocols and not as crisp algorithms. Furthermore, the available information for decision-making at this top level may be imprecise, qualitative, and incomplete as well. It follows that a high level of intelligence would be needed at the top level to interpret the available information on the system status and

to take appropriate “knowledge-based” actions for proper operation of the machine. In the machine, the knowledge base is represented as a set of fuzzy logic rules. This is in fact a collection of expert instructions on how to judge the product quality and what actions are needed to achieve a required performance from the machine under various operating conditions. The decision-making actions are carried out by matching the machine status with the knowledge base, using fuzzy logic, and particularly composition-based inference.

The high resolution information from the crisp sensors in the bottom layer of the machine has to be converted into a form that is compatible with the knowledge base in the top layer before a matching process can be executed. The intermediate layer of the hierarchy of Figure 9.8 accomplishes this task of information abstraction, where the sensory information from the bottom layer is preprocessed, interpreted, and represented for use in inference-making in the top layer. Each stream of sensory data from the bottom layer is filtered using a dedicated processor, which identifies trends and patterns of a set of representative parameters in these signals. The operations performed by the preprocessors may include averaging, statistical computations such as standard deviation and correlation, peak detection, pattern recognition, and computation of performance attributes such as rise time, damping ratio, and settling time of a machine component. Next, these parameters are evaluated according to appropriate performance criteria and transcribed into fuzzy quantities that are compatible with the fuzzy condition descriptors of the top-level knowledge base.

### 9.3.3 Information preprocessing

Next we will describe the procedures that are employed in processing the low-level sensory signals from the machine components so as to establish the performance status of the machine, and in particular, the quality of the processed fish. The specific sensory information that is processed in this manner includes visual images from the quality-monitoring CCD camera (secondary), servomotor position signals from the optical encoders, and the load profiles of the cutter blades.

#### 9.3.3.1 *Image preprocessing*

The main purpose of the secondary image processor of the machine is to extract context information from raw images, which will eventually lead to the extraction of high-level information such as the quality of processing and the quality of the end product. The problem of assessing the quality of a fish product is mostly esthetic in nature. It differs from the apparently similar problem of evaluating the quality of manufacture of an industrial component. In the latter case, the quality could be gauged through direct measurement of dimensions and to a lesser extent by the detection of surface defects. The quality of processed fish, in contrast, is not amenable to simple quantification based on a few dimensions, shapes or surface defects. At least

in theory, however, it could be considered as a complex combination of such features. In the sequel, several of these features are considered as qualitative or imprecise estimates that approximately indicate quality, and are consequently parameterized by means of fuzzy linguistic variables.

The top-level knowledge base of the hierarchical system of the machine will derive two context variables from the visual information; namely, the accuracy of cut and the quality of cut. The accuracy of cut can be considered deterministic since it is determined by the final error values of the cutter positioning system and the platform positioning system. In particular, the cutter position error is given by the measured difference between the gill position as gauged by the primary vision system and the actual position of cut as determined by the cutter positioning sensor. The quality of cut will be assessed based on the following geometric features:

- (1) Depth of the solid region of the cut section.
- (2) Smoothness of the boundary contour of the cut section.
- (3) Smoothness of the cut surface.

#### 9.3.3.2 Servomotor response preprocessing

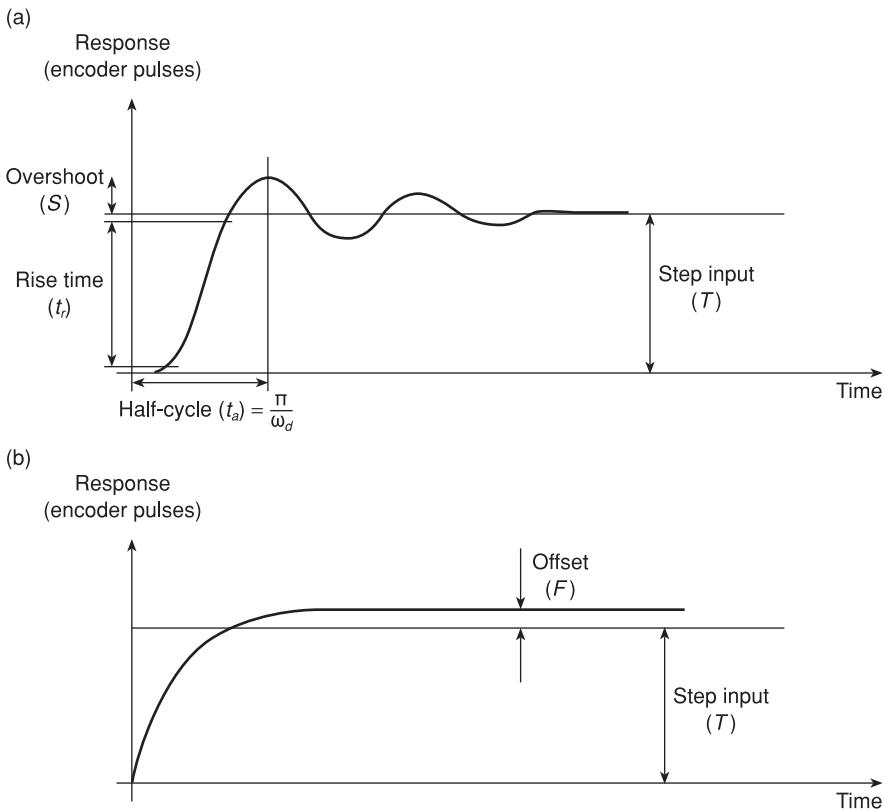
As indicated in Figure 9.7, the control and monitoring computer receives feedback from the two servomotors that drive the cutter assembly and the vertical positioning platform, in the form of position information via the GALIL DMC-620 two-axis controller. This information is filtered by the servomotor preprocessor and the following performance parameters, as applicable, are extracted:

- (1) Rise time (time to reach 95% of the step input).
- (2) Damping ratio.
- (3) Damped natural frequency (if underdamped).
- (4) Overshoot (if underdamped).
- (5) Offset (steady-state error).

Figure 9.9 shows the significance of these parameters for two typical cases of responses where one is underdamped and the other is overdamped. Since the required processing involves determination of maxima and minima, it is imperative that the stream of position data be filtered to remove any effect of noise and local disturbances. Consequently, the following second order binomial-filter function is used to smooth the servo-response data prior to further processing:

$$f(n) = \frac{1}{4}[x(n-1) + 2x(n) + x(n+1)] \quad (9.20)$$

The next step of processing is to locate the turning points of the response curves of Figure 9.9; specifically, the points  $a$ ,  $b$  and  $c$ , and the corresponding time instants  $t_a$ ,  $t_b$ , and  $t_c$ . This is done by means of an algorithm that searches



**Figure 9.9:** Typical parameters of servomotor response: (a) Underdamped case.  
(b) Overdamped case

for zero crossings of the first derivative of the step response. The rise time ( $T_r$ ) is then considered to be the time it takes for the response to reach 95% of the step input. The overshoot ( $S_p$ ) is calculated at the first peak of response, as a percentage of the step input. The steady-state error, which is computed as the percentage offset ( $F_p$ ), is determined by comparing the average of the last third of the response with the step input. With reference to the servomotor response in Figure 9.9, the following performance parameters can be estimated:

$$\text{Rise time } (T_r) = t_r \quad (9.21)$$

$$\text{Percentage overshoot } (S_p) = \left( \frac{S}{T} \right) \times 100\% \quad (9.22)$$

$$\text{Damped natural frequency } (\omega_d) = \frac{\pi}{t_a} \quad (9.23)$$

$$\text{Damping ratio } (\xi) = \sqrt{\frac{\ln^2(S/T)}{\pi^2 + \ln^2(S/T)}} \quad (9.24)$$

In equations (9.21) through (9.24), the dominant mode of the servomotor response is assumed to be second order – an assumption quite justifiable for the servomotors of the prototype machine. Equation (9.24) is obtained by rearranging the terms of the more familiar equation for the overshoot of a second order system, in terms of its damping ratio; thus,

$$\text{Overshoot } (S/T) = \exp - \left( \frac{\pi\xi}{\sqrt{1 - \xi^2}} \right) \quad (9.25)$$

In the next step of data preprocessing, the performance parameters computed by equations (9.21) through (9.24) are compared with those of a reference model. This model is considered as exhibiting the desired response, which is the performance specification, and is typically not a model of the actual physical system. For the servomotor system of the prototype machine, the following second order plant, with appropriate parameters, has been used as the reference model:

$$G_r = \frac{\omega_n^2}{(s^2 + 2\xi\omega_n s + \omega_n^2)} + G_d \quad (9.26)$$

where  $\omega_n$  and  $\xi$  are, respectively, the undamped natural frequency and the damping ratio. The term  $G_d$  represents external disturbances and can be regarded as a fixed offset for all practical purposes.

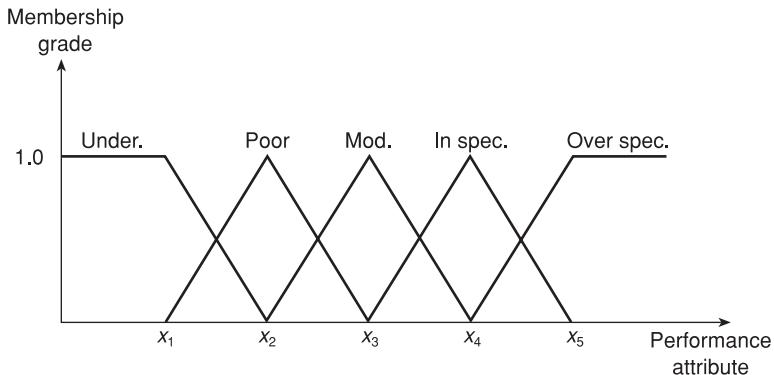
The knowledge base for servomotor tuning contains a set of rules prescribing what tuning actions should be taken when the actual response of the system deviates from the specifications. In the case of an intelligent tuner, this deviation is expressed in qualitative terms, signifying the experience of an expert. Accordingly, the context database of servomotor status uses fuzzy linguistic variables symbolizing the qualitative estimates of how the performance can deviate from the desired conditions. This is accomplished for rise time, percentage overshoot, and offset, first by comparing each actual performance parameter with the corresponding parameter of the reference model according to

$$\text{Index of deviation} = 1 - \frac{\text{Attribute of model}}{\text{Attribute of servomotor}} \quad (9.27)$$

and then using the resulting index to fuzzify each performance attribute into one of the five primary fuzzy sets (fuzzy states). For the performance parameters damped natural frequency and damping ratio, the index used is given by

$$\text{Index of deviation} = 1 - \frac{\text{Attribute of servomotor}}{\text{Attribute of model}} \quad (9.28)$$

Each index of deviation corresponds to one antecedent variable in the rule base and is defined such that when the actual performance is within



**Figure 9.10:** Multi-state membership function for each servomotor performance index

**Table 9.2:** Context database for servomotor performance

Context database of performance attributes	<i>rise_time, damped_natural_frequency, damping_ratio, overshoot, offset</i>
Primary term set for each performance deviation	<i>over_specification, in_specification, moderate, poor, unsatisfactory</i>

specification, the index is zero, and for the worst-case performance the index approaches unity. These numerical values are then fuzzified into membership functions with five primary fuzzy states, as shown in Figure 9.10. These data processing operations eventually result in a set of five fuzzy condition variables standing for the performance attributes as given in row 1 of Table 9.2, and a set of five primary fuzzy states (row 2) that each variable can assume. The database in fact contains two sets of such variables, one set for the cutter positioning servomotor and the other for the platform positioning servomotor.

### 9.3.3.3 Cutter load preprocessing

The purpose of the cutter-load-profile preprocessor is to extract useful high-level information pertaining to the performance of the cutting action itself. To this end, the speeds of the top and the bottom blades of the cutter unit are monitored through speed sensors and associated circuitry. The goal is to eventually derive from these data such context information as peak and average load on the cutter motors, symmetry of feeding of fish into the cutter, and the presence of any undesirable lateral movement (slip) of fish while cutting takes place.

The speeds of the induction motors, which drive the cutter blades, are first converted into the corresponding *slip* values and then to the percentage nominal loads, using

$$\text{Measured slip } (s_m) = \frac{\text{Synchronous speed} - \text{Measured shaft speed}}{\text{Synchronous speed}} \quad (9.29)$$

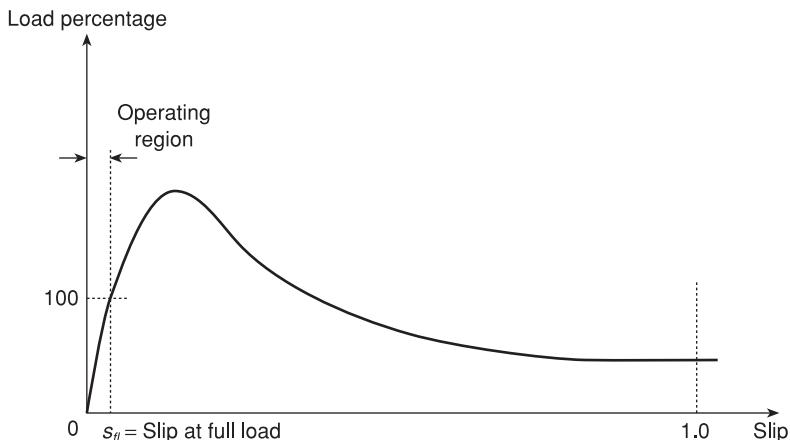


Figure 9.11: Load percentage versus slip for the induction motors

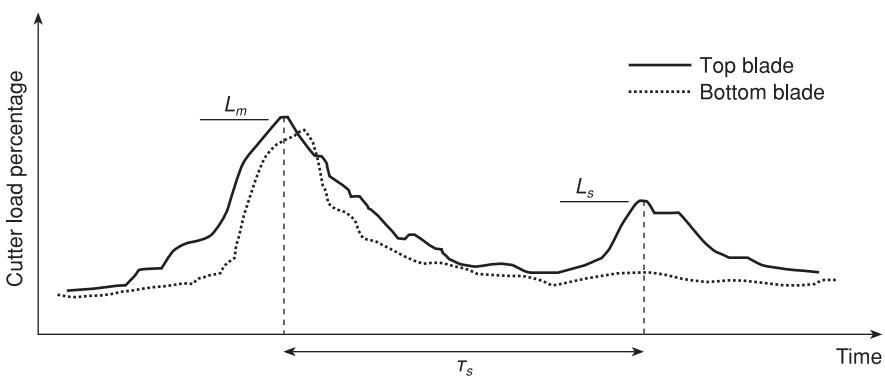


Figure 9.12: Typical load profiles of the two cutter blades

where the synchronous speed for the induction motors of the machine is 3600 rpm.

$$\text{Percent load } (L_p) = \frac{\text{Measured slip } (s_m)}{\text{Slip at full load } (s_f)} \times 100\% \quad (9.30)$$

Equation (9.30) assumes that the motor operates in the linear region of its load-slip curve, as indicated in Figure 9.11. This assumption holds true in general for all commercially available induction motors of type "D", such as those used in the machine.

A typical load profile obtained after these preliminary data processing operations is shown in Figure 9.12, where the load on each of the two cutter blades during a typical cutting operation is plotted with respect to time. The load-profile data contain interference or noise created by the frequency-to-analog conversion, which usually appears as easily-visible ripples on the load profile. The 4th order binomial filter given by

$$f(n) = \frac{1}{16} [x(n-2) + 4x(n-1) + 6x(n) + 4x(n+1) + x(n+2)] \quad (9.31)$$

where  $x(i)$  is the data sample at the  $i$ -th instant, is used in order to screen out the effect of these ripples before further processing. Now, the following parameters can be conveniently measured from the two individual profiles: peak load, average load, magnitude of secondary peaks, if present, and their times of occurrence relative to the first peak.

The average load of each curve over a single cutting cycle signifies the average cutting load on the induction motors. The difference in magnitudes between the two peak loads of the top and the bottom blades is typically an indication of the degree of asymmetry in feeding fish into the cutter. The presence of secondary peaks may indicate slipping of fish during cutting, and hence, improper holding by the gripper. A secondary peak that is spaced closely to the primary peak is a common feature in a majority of cut profiles, indicating the load variations when the blades are cutting through bones rather than meat. This feature, however, can be set apart from the undesirable secondary peaks due to improper holding simply by timing the secondary peaks with respect to the primary one and then ignoring the ones that are sufficiently close to the primary peak. It follows that the timing of secondary peaks is crucial in recognizing the presence of improper holding.

For the purpose of context generation, the non-dimensional indices given by

$$\text{Cut load percentage } (L_{pc}) = \frac{1}{2}(L_{mt} + L_{mb}) \quad (9.32)$$

$$\text{Index of asymmetry } (I_a) = \frac{\left| L_{mt} - L_{mb} \right|}{\frac{1}{2}(L_{mt} + L_{mb})} \quad (9.33)$$

$$\text{Secondary peak load index } (I_{ls}) = \frac{2L_s}{(L_{mt} + L_{mb})} \quad (9.34)$$

$$\text{Index of secondary-peak time } (I_{ts}) = \frac{\tau_s}{T} \quad (9.35)$$

are computed using the parameters illustrated in Figure 9.12. The subscripts  $t$  and  $b$  denote the “top” and the “bottom” blades, respectively, and  $1/T$  is the feed rate of fish on the conveyor. The indices for secondary peak load ( $I_{ls}$ ) and time of secondary peak load ( $I_{ts}$ ) are computed for the two blades, separately. These indices are fuzzified using the membership function shown in Figure 9.13, and then the following rule base submodule is used in conjunction with the compositional rule of inference, to generate the membership functions of *Slip\_at\_top\_blade* and *Slip\_at\_bottom\_blade*:

- If secondary peak load ( $I_{ls}$ ) is present and secondary peak load index ( $I_{ls}$ ) is medium, then *slip* is moderate.

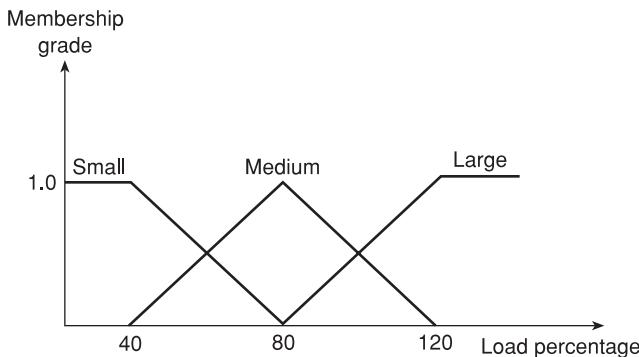


Figure 9.13: Membership function for the cutter load context

- Else if secondary peak load ( $I_{ls}$ ) is present and secondary peak load index ( $I_{ls}$ ) is high then *slip* is large.
- Else *slip* is small.

The knowledge base of the machine supervisory system has an antecedent variable representing the slip at the blades. In order to generate the context of this composite slip, the two membership states of *Slip\_at\_top\_blade* and *Slip\_at\_bottom\_blade* are composed further by the following rule base module:

- If slip at top blade is large and slip at bottom blade is large then *composite\_slip* is large.
- Else if slip at top blade is moderate or slip at bottom blade is moderate then *composite\_slip* is moderate.
- Else if slip at top blade is small and slip at bottom blade is small then *composite\_slip* is small.

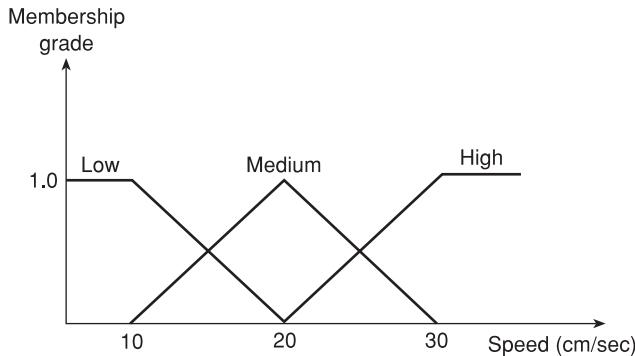
These preprocessing operations eventually result in a set of three fuzzy-status variables, which stand for the cutter performance as depicted in column 1 of Table 9.3. Column 2 shows the primary fuzzy terms (fuzzy states) that each variable can assume.

Table 9.3: Context database for cutter performance

Context variable of cutter performance	Primary term set for each variable
<i>Percent_cut_load</i>	Small, Medium, Large
<i>Index_of_asymmetry</i>	Small, Medium, Large
<i>Composite_slip</i>	Small, Moderate, Large

**Table 9.4:** Context database for conveyor performance

Context of conveyor performance	Primary term set
Conveyor_speed Speed_fluctuations	Low, Medium, High Small, Medium, Large

**Figure 9.14:** Membership functions for conveyor performance

### 9.3.3.4 Conveyor speed preprocessing

The purpose of the preprocessor for conveyor speed profile is to extract high-level information pertaining to the performance of the drive unit of the conveyor during operation of the machine. Since the conveyor speed is measured directly through dedicated hardware, the degree of preprocessing required on this channel of information is minimal. The following two items of context data are extracted from the conveyor speed samples:

$$\text{Average speed } (V_{av}) = \frac{1}{n} \sum_{k=1}^n v_{kt} \quad (9.36)$$

$$\text{Speed fluctuations } (V_\Delta) = \frac{\max_{k=1}^n (v_{kt}) - \min_{k=1}^n (v_{kt})}{V_{av}} \quad (9.37)$$

which are acquired during each cycle of fish processing; where  $T = nt$  is the cycle time of processing. Table 9.4 shows the resulting fuzzy variables which represent the status of the conveyor. The input space is partitioned into primary fuzzy sets that are named in column 2 of Table 9.4, and shown in Figure 9.14.

### 9.3.4 Knowledge-based decision-making

The knowledge base of a fuzzy control system is primarily comprised of a linguistic rule base. The inference-making mechanism makes up the production

engine, which carries out reasoning using the knowledge base, and intermediates the relation between the database (input space) and the inferences (output space). The reasoning process is triggered by the context data in the database, and may be thought of as an “approximate” matching process of the “context” to the condition parts of the rules. This is the basis of the so-called “approximated reasoning.” In the context of a hierarchical structure, the database is essentially a conceptual abstraction of the current status of the process, expressed with respect to the corresponding fuzzy linguistic variables. The derivation of such a database through the use of information preprocessors was discussed previously. Now we will describe the development of the rule base which, by acting on the current context, will assess the performance of the machine and the quality of the finished product. The rule base will also infer remedial actions where necessary.

#### 9.3.4.1 Knowledge acquisition

A fuzzy rule base is characterized by a collection of linguistic statements of expert knowledge that is usually expressed in the form of *if-then* rules. In fuzzy logic, such rules are implemented by fuzzy conditional statements. In the fields of process control and system tuning, there are at least four possible sources for the derivation of fuzzy control rules. A brief discussion of these four methods is given now, with particular attention to their usefulness in relation to the present industrial application.

***Use of expert experience and control engineering knowledge:*** This method treats the process states as the antecedent variables of fuzzy control rules and the process control inputs as the consequent variables. The most common formulation involves an introspective verbalization of human expertise. Another approach includes interrogation of experienced operators using a carefully designed questionnaire. This will allow one to form a system of fuzzy control rules for a particular application domain. The present application has a rule base module for servomotor tuning, which has been formulated entirely on the basis of this method, particularly by employing the control engineering knowledge of servomotor tuning. Experimentation was initially carried out on the servomotors to acquire experience on the effect of specific control parameters over the motor performance, and this experience was then formulated as a fuzzy linguistic rule base.

***Use of control actions of an operator:*** In many types of industrial human-machine control systems, the input-output relations are not tractable with sufficient precision to enable the utilization of classical control theory together with analytical modeling, experimental modeling, and simulation. Many of the subsystems in the fish processing machine are prime examples of this type of intractable control system. Skilled human operators can, however, control such systems quite successfully without using an explicit analytical or quantitative model. It has been shown that a human operator in effect employs a set of fuzzy if-then rules to control a process. The rule base

modules used for tuning such parameters as the conveyor speed and the hold-down force have been developed on the basis of this method. Due to the lack of prior experience on certain aspects of the automated machine, some rules were tested on a trial-and-error basis and then this experience was utilized for fine-tuning of the rule base.

**An approach based on learning:** This approach refers to the ability to create fuzzy control rules and modify them automatically based on previous experience. Such systems are usually organized in a hierarchy consisting of two rule bases. The first one is a typical rule base of a fuzzy control system while the second one is a set of *meta-rules* that exhibits a human-like learning ability to create and modify a general rule base, depending on the desired performance of the system. Although this method is not used in our machine, it is an attractive proposition as a possible enhancement to the rule base. When modifying the rule base for ascertaining the quality of a finished product, for example, a self-learning rule base would be of significant use due to the subjective nature of the particular task involved.

**Use of a fuzzy model for the process:** If a linguistic description of the dynamic characteristics of a controlled process is available, then it may be viewed as a fuzzy model of the process. Based on this fuzzy model, one can generate a set of fuzzy control rules for attaining optimal performance of the dynamic system. In practice, this approach tends to be somewhat more complicated, and a systematic design methodology has not yet been fully developed. This approach, however, has been shown to deliver good performance and reliability. Due to the lack of such linguistic characterization for the processes of our machine, this method has not been used in the present application.

Two types of fuzzy control rules, namely, state evaluation fuzzy control rules and predictive (object evaluation) fuzzy control rules, are in use in the design of fuzzy controllers. The state evaluation type, in the case of multi-input-single-output (MISO) systems, is characterized as a collection of rules of the form:

Rule 1: If  $a$  is  $A_1$  and  $b$  is  $B_1 \dots$ , then  $z$  is  $Z_1$

:                   :                   :

Rule  $n$ : If  $a$  is  $A_n$  and  $b$  is  $B_n \dots$ , then  $z$  is  $Z_n$

where  $a, b, \dots$ , and  $z$  are linguistic variables representing the process state variables and the control variable, respectively.  $A_i, B_i, \dots$ , and  $Z_i$  are the linguistic values that the fuzzy variables  $a, b, \dots$ , and  $z$  can assume in the universes of discourse  $U, V, \dots$ , and  $W$ , respectively, for  $i = 1, 2, \dots, n$ .

The predictive fuzzy control is a fuzzy algorithm that predicts present and future control actions and evaluates control objectives. A typical rule can be described as

Rule  $i$ : If ( $u$  is  $U_i \rightarrow (a$  is  $A_i$  and  $b$  is  $B_i))$  then  $u$  is  $U_i$

where  $a$  and  $b$  are performance indices for evaluation of the  $i$ -th rule. The most likely control rule is selected through predicting the result ( $a$  and  $b$ ) corresponding to every control command  $U_i$ .

The rule base of the prototype machine has been implemented entirely in the form of state-evaluation fuzzy control rules. The predictive format could not be used in the system development stage due to lack of complete knowledge of system dynamics. Such advance knowledge, even though counterproductive in the present application, is vital in formulating the performance indices in a predictive scheme.

#### 9.3.4.2 Decision-making

The inference engine is an essential mechanism for knowledge-based control and it is solely responsible for generating inferences corresponding to the current context of the machine. The database, which contains the current context, and the rule base, which holds the domain-specific knowledge, are two important components that are used by the inference engine. The absence of a chaining mechanism will simplify the process of inference-making under the context of fuzzy logic. Unlike the conventional production engines where forward or backward chaining is essential in their inference mechanism, fuzzy logic employs the *Compositional Rule of Inference* as the inference-making scheme, where every rule has a certain contribution towards the final outcome. The strength of this contribution is determined by the relative strength of the membership functions involved.

Every rule in the knowledge base of a fuzzy system describes a relation between the antecedent and consequent variables of the particular rule. The degree of strength of this relation is fuzzy and can be expressed in the form of a relational table. If the knowledge base contains a total of  $N$  antecedent (condition) variables and  $M$  consequent (action) variables, then the entire rule base can be expressed by a relational table of dimension  $(N + M)$ . In practice, however, this approach has many drawbacks such as large processing time and difficulty of maintenance. Instead, since the whole rule base could be partitioned into several multi-input-single-output modules, each such module could be designated by a relational table of dimension  $(n + 1)$ , where  $n$  is the dimension of the input space; i.e., the total number of different antecedent variables that contribute towards a single output variable. The relational table is constructed by applying fuzzy composition to the membership functions of the variables in the rule base. It follows that an elementary implementation of a fuzzy system is a fuzzy variable, which is represented in the machine system by a two-dimensional array of  $(p \times d)$  elements, where  $p$  is the number of primary fuzzy terms (partitions) or fuzzy states of the variable and  $d$  is the number of discretization levels. The number of fuzzy states,  $p$ , defined over a particular universe of discourse, depends on the level of granularity of the desired control. Usually, in practice,  $p$  is assigned a number among 2, 3, 5, and 7. The number of discretization levels  $d$  essentially refers to quantization of the continuous universe of a fuzzy variable, for digital computation.

**Table 9.5:** Cut\_load\_mem[i][j] – representation of the fuzzy variable “cut\_load”

	0	1	2	3	4	5	6	7	8	9	10
Low	1.0	1.0	1.0	0.7	0.3	0.0	0.0	0.0	0.0	0.0	0.0
Med	0.0	0.0	0.0	0.3	0.7	1.0	0.7	0.3	0.0	0.0	0.0
High	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.7	1.0	1.0	1.0

The particular choice of quantization level has a direct influence on how fine a control can be achieved. Consequently,  $d$  should be large enough to provide an adequate approximation and yet small enough to save storage requirements. For example, if every fuzzy variable has an identical level of discretization of  $d$ , then the size of the relational table of a rule module would be  $d^{n+1}$ , which could become extremely large for typical values of  $d$ . The machine uses a discretization level of 11 for most of its fuzzy variables. Consider, as an example, the representation of the fuzzy variable *cut\_load*, where  $p = 3$  (low, medium, high) and  $d = 11$ . Table 9.5 shows how the membership functions that are illustrated in Figure 9.13 for the variable *cut\_load* are stored as a  $3 \times 11$  array of elements.

Once the membership functions for all antecedent variables and for the single consequent variable of the rule base module are defined, the relational table is constructed as explained by the following example:

Consider a rule base fragment with two antecedent variables, *cut\_load* and *slip*, and a single consequent variable, *speed*, as given by

Rule 1: IF (*cut\_load=low*) AND (*slip=small*) THEN (*speed=pos\_large*)  
 Rule 2: IF (*cut\_load=med*) AND (*slip=small*) THEN (*speed=pos\_small*)  
 Rule 3: IF (*cut\_load=high*) AND (*slip=small*) THEN (*speed=neg\_small*)  
 :                           :  
 :                           :

First, the primary term sets of each fuzzy variable are enumerated in such a manner that an array reference would appear linguistically equivalent to the corresponding clause in the corresponding rule. For example, the membership value for the clause *cut\_load=low* could be referenced as *cut\_load\_mem[low]*.

```
enum {low, medium, high} cut_load_terms;
enum {small, moderate, large} split_terms;
enum {neg_large, neg_small, zero, pos_small, pos_large} speed_terms;
```

Next, the relation table is built by applying every rule to all possible combinations of input values. That is, for a single instance of input values, the corresponding rule-firing strengths are calculated in accordance with the fuzzy norms of AND, OR, NOT, and IF-THEN, for every rule. The maximum value of these firing strengths is then assigned to the corresponding

instance of input combination. This procedure is repeated for all possible instances in the input space as follows:

```

for (cut_load = 0; cut_load < d; cut_load++)
    for (slip = 0; slip < d; slip++)
        for (speed = 0; speed < d; speed++)
{
    rule1 = minimum (cut_load_mem [low] [cut_load],
    slip_mem [small] [slip],
    speed_mem [pos_large] [speed]);
    rule2 = minimum (cut_load_mem [medium] [cut_load],
    slip_mem [small] [slip],
    speed_mem [pos_small] [speed]);
    rule1 = minimum (cut_load_mem [high] [cut_load],
    slip_mem [small] [slip],
    speed_mem [neg_small] [speed]);
    :
    :
    rule_speed [cut_load] [slip] [speed]
= maximum (rule1, rule2, rule3, .....);
}

```

where *rule\_speed* is the three-dimensional relational table which connects the two-dimensional input space (*cut\_load*  $\times$  *slip*) to the single output (*speed*).

In real-world control applications typical inputs are fuzzy singletons. Nevertheless, the present realization of the machine system supports the more general, fuzzy compositional rule of inference. Specifically, the inferencing mechanism in the machine assumes the inputs to be fuzzy variables with a broad support set rather than a single element.

The first step of inference-making is to form the *join* (or intersection) of the input values. For the input example that was given previously, one can write,

```

for (cut_load = 0; cut_load < d; cut_load++)
    for (slip = 0; slip < d; slip++)
{
    join [cut_load] [slip] = minimum (in_cut_load [cut_load],
    in_slip [slip]);
}

```

where *join* is the two-dimensional input vector in the universe of discourse (*cut\_load*  $\times$  *slip*), and *in\_cut\_load* and *in\_slip* are the membership functions corresponding to the input (condition) variables *cut\_load* and *slip*, respectively. Then we can apply the compositional rule of inference to the input vector *join* and relational table *rule\_speed*, to obtain the fuzzy inference of *speed*, as follows:

```

for (cut_load = 0; cut_load < d; cut_load++)
    for (slip = 0; slip < d; slip++)
        for (speed = 0; speed < d; speed++)
{
    min = minimum (rule_speed [cut_load] [slip] [speed],
join [cut_load] [slip]);
    out_speed [speed]
    = maximum (out_speed [speed], min);
}

```

where *out\_speed* is the inference of *speed* expressed in terms of a membership function with the discretization level *d*. Depending on the level of abstraction, this inference may be either defuzzified and used as a control variable or it may be used as a fuzzy input to a higher-level knowledge base of the system hierarchy.

The machine system implements the centroid method of defuzzification for all its outputs. This method takes the center of area of the membership function after assigning suitable weights to all discretized points in the corresponding universe of discourse; thus,

$$\text{Output} = \frac{\sum_{i=0}^d (\text{out\_speed}[i] \times \text{weight\_speed}[i])}{\sum_{i=0}^d \text{out\_speed}[i]} \quad (9.38)$$

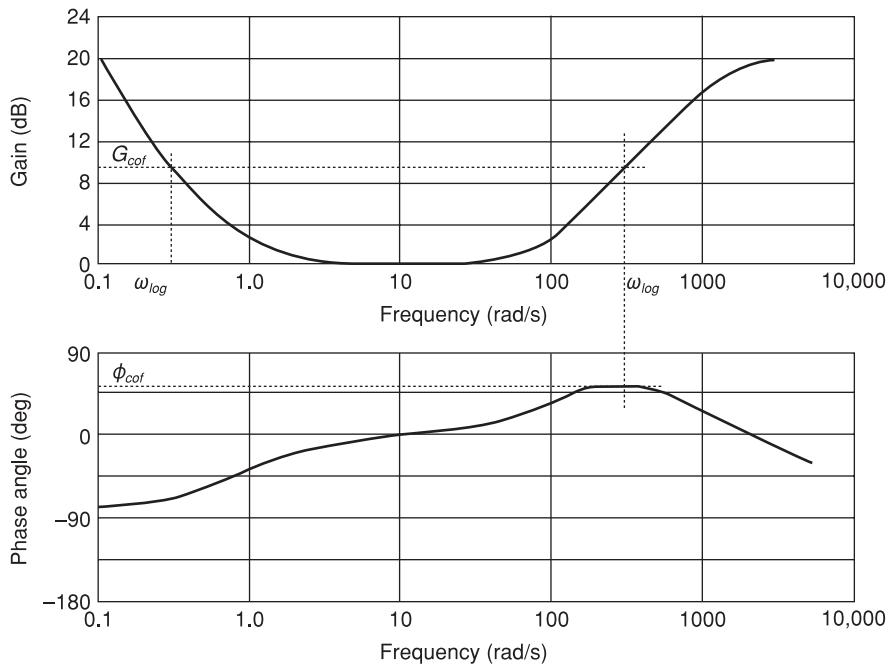
where *weight\_speed* is a weighting function defined at every discretization point of the membership function *out\_speed*. As an example, since the membership function *out\_speed* refers to the change of speed in either positive or negative sense, the weighting function *weight\_speed* takes the form

$$\text{weight\_speed} = [-5, \dots, -1, 0, 1, \dots, 5]$$

which, in a physical implementation, should have the engineering dimensions of the output variable. The control variables will be further scaled by a sensitivity factor before they are transmitted to the respective controllers.

#### 9.3.4.3 Servo tuning

Tuning under the control environment of DMC-600 has been studied previously. The tuning rule base was developed by applying this control-engineering knowledge to the machine in an on line basis. The antecedent variables (i.e., the condition or status of the servomotor) and the fuzzy values that each variable can assume are given in Table 9.6. The consequent (i.e., action) variables represent the design attributes of the controller that have to be tuned in order to achieve a desired performance. These design attributes relate to the frequency domain specifications of a lead compensator as illustrated



**Figure 9.15:** Frequency domain specifications (design parameters) of a lead compensator as applicable to the rule base for servomotor tuning

**Table 9.6:** Consequent variables for servomotor tuning

Variable	Description	Notation
$\omega_{cog}$	Frequency at maximum phase lead of compensator	FRCOG
$\phi_{cof}$	Maximum phase lead of compensator	PHCOF
$G_{cof}$	Compensator gain at maximum phase lead	CNCOF
$\omega_{lof}$	Integrator frequency for a gain of $G_{cof}$	LFCOG

in Figure 9.15, along with a pure integral controller in parallel. Each of these consequent variables, as listed in Table 9.6, can assume one of five possible fuzzy states that are defined over a normalized output space  $[-1.0, +1.0]$ , as shown in Figure 9.16.

The tuning rule base is given in Table 9.7. Those consequent variables that do not appear in some of the rules in this rule base must be considered as not contributing to the tuning of the servomotor controller. In other words, they all take the fuzzy value  $ze$  (zero) in the output space.

The design attributes are updated by the incremental values that the rule base infers. The resulting values are then mapped to the controller parameters of the frequency domain transfer function of a lead compensator

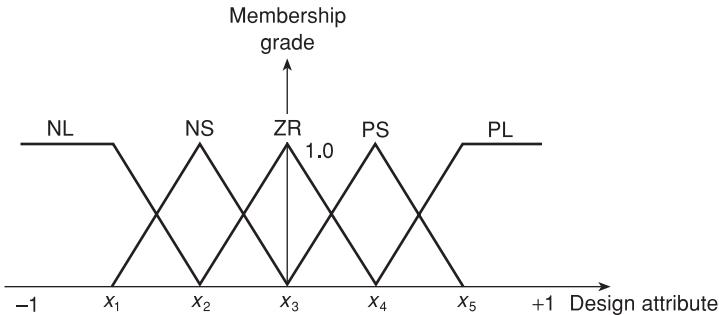


Figure 9.16: Output space partitioning for the rule base of servomotor tuning

Table 9.7: The rule base for servomotor tuning

If	<i>Rise_time</i> is <i>Unsatis</i>	then	<i>frcog</i> is <i>nl</i> & <i>phcof</i> is <i>nl</i> & <i>gncof</i> is <i>pl</i>
else if	<i>Rise_time</i> is <i>Poor</i>	then	<i>frcog</i> is <i>ns</i> & <i>phcof</i> is <i>ns</i> & <i>gncof</i> is <i>ps</i>
else if	<i>Rise_time</i> is <i>Moderate</i>	then	<i>frcog</i> is <i>ns</i> & <i>phcof</i> is <i>ns</i> & <i>gncof</i> is <i>ps</i>
else if	<i>Rise_time</i> is <i>In_spec</i>	then	<i>frcog</i> is <i>ze</i> & <i>phcof</i> is <i>ze</i> & <i>gncof</i> is <i>ze</i>
else if	<i>Rise_time</i> is <i>Over_spec</i> .	then	<i>frcog</i> is <i>ps</i> & <i>phcof</i> is <i>ps</i> & <i>gncof</i> is <i>ns</i>
If	<i>Damping_ratio</i> is <i>Unsatis.</i>	then	<i>frcog</i> is <i>nl</i> & <i>phcof</i> is <i>pl</i> & <i>gncof</i> is <i>pl</i>
else if	<i>Damping_ratio</i> is <i>Poor</i>	then	<i>frcog</i> is <i>ns</i> & <i>phcof</i> is <i>ps</i> & <i>gncof</i> is <i>ps</i>
else if	<i>Damping_ratio</i> is <i>Moderate</i>	then	<i>frcog</i> is <i>ns</i> & <i>phcof</i> is <i>ps</i> & <i>gncof</i> is <i>ps</i>
else if	<i>Damping_ratio</i> is <i>In_spec.</i>	then	<i>frcog</i> is <i>ze</i> & <i>phcof</i> is <i>ze</i> & <i>gncof</i> is <i>ze</i>
else if	<i>Damping_ratio</i> is <i>Over_spec.</i>	then	<i>frcog</i> is <i>ps</i> & <i>phcof</i> is <i>ns</i> & <i>gncof</i> is <i>ns</i>
If	<i>Overshoot</i> is <i>Unsatis</i>	then	<i>frcog</i> is <i>nl</i> & <i>phcof</i> is <i>pl</i> & <i>gncof</i> is <i>pl</i>
else if	<i>Overshoot</i> is <i>Poor</i>	then	<i>frcog</i> is <i>ns</i> & <i>phcof</i> is <i>ps</i> & <i>gncof</i> is <i>ps</i>
else if	<i>Overshoot</i> is <i>Moderate</i>	then	<i>frcog</i> is <i>ns</i> & <i>phcof</i> is <i>ps</i> & <i>gncof</i> is <i>ps</i>
else if	<i>Overshoot</i> is <i>In_spec.</i>	then	<i>frcog</i> is <i>ze</i> & <i>phcof</i> is <i>ze</i> & <i>gncof</i> is <i>ze</i>
else if	<i>Overshoot</i> is <i>Over_spec.</i>	then	<i>frcog</i> is <i>ps</i> & <i>phcof</i> is <i>ns</i> & <i>gncof</i> is <i>ns</i>
If	<i>Offset</i> is <i>Unsatis.</i>	then	<i>gncof</i> is <i>pl</i> & <i>lfcog</i> is <i>pl</i>
else if	<i>Offset</i> is <i>Poor</i>	then	<i>gncof</i> is <i>pl</i> & <i>lfcog</i> is <i>ps</i>
else if	<i>Offset</i> is <i>Moderate</i>	then	<i>gncof</i> is <i>ps</i> & <i>lfcog</i> is <i>ps</i>
else if	<i>Offset</i> is <i>In_spec.</i>	then	<i>gncof</i> is <i>ps</i> & <i>lfcog</i> is <i>ze</i>
else if	<i>Offset</i> is <i>Over_spec.</i>	then	<i>gncof</i> is <i>ze</i> & <i>lfcog</i> is <i>ns</i>
If	<i>Dmp_nat_freq.</i> is <i>Unsatis.</i>	then	<i>gncof</i> is <i>pl</i>
else if	<i>Dmp_nat_freq.</i> is <i>Poor</i>	then	<i>gncof</i> is <i>ps</i>
else if	<i>Dmp_nat_freq.</i> is <i>Moderate</i>	then	<i>gncof</i> is <i>ps</i>
else if	<i>Dmp_nat_freq.</i> is <i>In_spec.</i>	then	<i>gncof</i> is <i>ze</i>
else if	<i>Dmp_nat_freq.</i> is <i>Over_spec.</i>	then	<i>gncof</i> is <i>ns</i>

$$G(j\omega) = K \left( \frac{1 + A \cdot j\omega}{1 + \alpha A \cdot j\omega} \right), \quad 0 < \alpha < 1 \quad (9.39)$$

where  $K$  is the gain parameter of the compensator, while  $A$  and  $\alpha A$  are time constants associated with the lead compensator. The following equations complete the mapping procedure from design attributes to the controller parameters:

$$\alpha = \frac{1 - \sin(\phi_{cof})}{1 + \sin(\phi_{cof})} \quad (9.40)$$

$$A = \frac{1}{\omega_{cog}\sqrt{\alpha}} \quad (9.41)$$

$$K = \sqrt{\alpha} |G_{cof}| \quad (9.42)$$

according to the definitions given in Table 9.6. In addition, the controller has an integrator branch with the transfer function

$$G_i(j\omega) = \frac{K_i}{j\omega} \quad (9.43)$$

According to Table 9.6, the gain parameter  $K_i$  is given by

$$K_i = \omega_{lof} |G_{cof}| \quad (9.44)$$

The above procedure of parameter updating is repeated for both the cutter servomotor and the platform servomotor. The rule base for servomotor tuning operates independently on the separate context databases of the two servomotors, producing inferences for tuning each servomotor independently.

#### 9.3.4.4 Product quality assessment

Quality of the finished product from the machine is assessed primarily on the basis of visual context that is determined through preprocessing of image data from the secondary CCD camera. Specifically, the features that indicate the quality are (a) Accuracy of cut (*Accuracy*), (b) Depth of the solid region of the cut section (*Depth*), (c) Smoothness of the boundary contour of the cut section (*Contour*), and (d) Smoothness of the cut surface (*Surface*). The inference of the associated rule base is a single fuzzy variable indicating the quality of the finished product, as one of two possible fuzzy outcomes of either *Acceptable* or *Unacceptable*, as represented by the membership functions in Figure 9.17. The rule base has been devised to mimic the judgment of a human expert who assesses the quality of fish products by visual inspection.

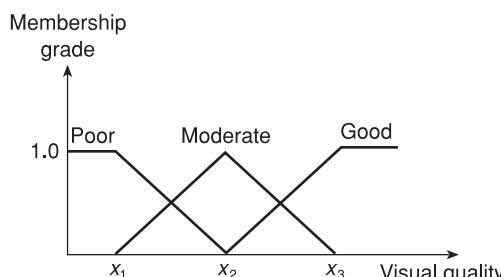


Figure 9.17: Output space partitioning for the quality assessment rule base

**Table 9.8: Rule base for assessment of product quality**

- 
- IF *Accuracy* is *Unacceptable* and *Depth* is *Any* and *Contour* is *Any* and *Surface* is *Any* THEN *Quality* is *Unacceptable*.
  - IF *Accuracy* is *Acceptable* and *Depth* is *Good* and *Contour* is *Good* and *Surface* is *Good* THEN *Quality* is *Acceptable*.
  - IF *Accuracy* is *Acceptable* and *Depth* is *Good* and *Contour* is *Moderate* and *Surface* is *Good* THEN *Quality* is *Acceptable*.
  - IF *Accuracy* is *Acceptable* and *Depth* is *Moderate* and *Contour* is *Moderate* and *Surface* is *Moderate* THEN *Quality* is *Acceptable*.
  - IF *Accuracy* is *Acceptable* and *Depth* is *Poor* and *Contour* is *Any* and *Surface* is *Any* THEN *Quality* is *Unacceptable*.
  - IF *Accuracy* is *Acceptable* and *Depth* is *Any* and *Contour* is *Poor* and *Surface* is *Any* THEN *Quality* is *Unacceptable*.
  - IF *Accuracy* is *Acceptable* and *Depth* is *Any* and *Contour* is *Any* and *Surface* is *Poor* THEN *Quality* is *Unacceptable*.
- 

The rule base module utilized in the machine for this purpose is shown in Table 9.8. In this rule base, the term *Any* in the antecedent variables refers to the conjunction of all possible values in that variable. For example, *Depth* is *Any* is equivalent to the conjunction of *Depth* is *Poor* or *Depth* is *Moderate* or *Depth* is *Good*.

#### 9.3.4.5 Machine tuning

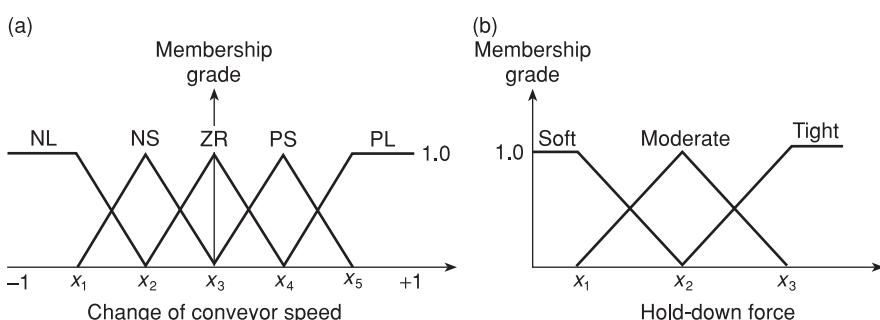
The rule base for machine tuning operates on the context derived from the low-level sensory measurements of conveyor speed, cutter load, and information on poor holding (slip) at the cutter. This database consists of the following fuzzy antecedent variables: (a) percentage cutter load (*Cutload*), (b) composite slip at cutter (*Slip*), (c) average conveyor speed (*Speed*), and (d) conveyor speed fluctuations (*Fluctuations*). The consequent or action variables are the tuning inferences: (a) the change in conveyor speed (*Speed change*), and (b) the degree of grasping (*Hold-down*). It should be noted that while the inference on conveyor speed is incremental, the inference on grasping, i.e., the hold-down force, is absolute. In addition, the absolute speed of the conveyor is limited by a pair of upper and lower bounds that are set through the defuzzification algorithm. Table 9.9 shows the rule base module that has been adopted for tuning of the machine. Although this table depicts the rule base as a MIMO system, the machine implements the two inferences individually in the form of two independent MISO systems. Figure 9.18 shows the partitioning of the two fuzzy output spaces corresponding to the respective inferences of change of conveyor speed and hold-down force.

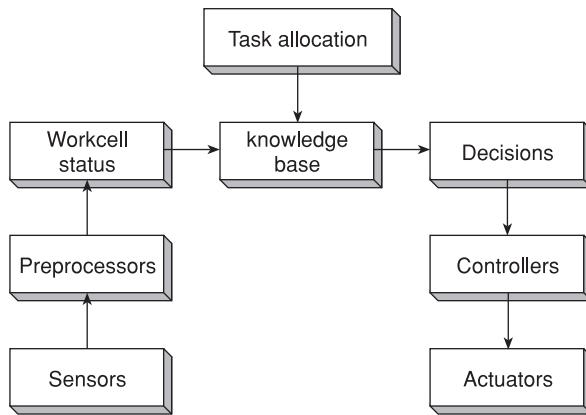
#### 9.3.5 System implementation

The hardware indicated in Figure 9.7 has been integrated into a complete machine. Software for both low-level and high-level functions of the system has been developed and implemented. Associated functions involve sensor

**Table 9.9:** Rule base for machine tuning

- IF Cutload is Low and Slip is Small and Fluctuations is Not Large and Speed is Low THEN Speed change is Pos. Large and Hold-down is Soft.
- IF Cutload is Low and Slip is Small and Fluctuations is Not Large and Speed is Medium THEN Speed change is Pos. Small and Hold-down is Soft.
- IF Cutload is Low and Slip is Small and Fluctuations is Not Large and Speed is High THEN Speed change is No Change and Hold-down is Soft.
- IF Cutload is Medium and Slip is Small and Fluctuations is Not Large and Speed is Low THEN Speed change is Pos. Small and Hold-down is Soft.
- IF Cutload is Medium and Slip is Small and Fluctuations is Not Large and Speed is High THEN Speed change is Pos. Small and Hold-down is Soft.
- IF Cutload is Medium and Slip is Small and Fluctuations is Not Large and Speed is High THEN Speed change is No Change and Hold-down is Soft.
- IF Cutload is High and Slip is Small and Fluctuations is Not Large and Speed is Low THEN Speed change is No Change and Hold-down is Moderate.
- IF Cutload is High and Slip is Small and Fluctuations is Not Large and Speed is Medium THEN Speed change is Neg. Small and Hold-down is Moderate.
- IF Cutload is High and Slip is Small and Fluctuations is Not Large and Speed is High THEN Speed change is Neg. Large and Hold-down is Moderate.
- IF Cutload is High and Slip is Small and Fluctuations is Not Large and Speed is Medium THEN Speed change is Neg. Small and Hold-down is Moderate.
- IF Cutload is Any and Slip is Moderate and Fluctuations is Not Large and Speed is Medium THEN Speed change is Neg. Small and Hold-down is Moderate.
- IF Cutload is Any and Slip is Moderate and Fluctuations is Not Large and Speed is Large THEN Speed change is Neg. Large and Hold-down is Moderate.
- IF Cutload is Any and Slip is Large and Fluctuations is Not Large and Speed is Any THEN Speed change is Neg. Large and Hold-down is Tight.
- IF Cutload is Any and Slip is Any and Fluctuations is Moderate and Speed is Medium THEN Speed change is No Change and Hold-down is Moderate.
- IF Cutload is Any and Slip is Any and Fluctuations is Moderate and Speed is Large THEN Speed change is Neg. Small and Hold-down is Moderate.
- IF Cutload is Any and Slip is Any and Fluctuation is Large and Speed is Any THEN Speed change is Neg. Small and Hold-down is Tight.

**Figure 9.18:** Output space partitioning for the rule base of machine tuning:  
(a) For conveyor speed. (b) For hold-down force



**Figure 9.19:** Software modules of the machine

reading and generation of low-level control actions, component synchronization, signal preprocessing, supervisory decision-making, and execution of tuning actions.

The control hierarchy has been realized in several C-language-programmed modules. Selection of the language C for realization of the machine software was particularly influenced by factors such as:

- Availability of low-level machine instructions in C facilitates easy interfacing with the low-level controllers.
- Fast speed of execution and the availability of support for a graphical user interface.

#### 9.3.5.1 System modules

The modular development of software closely resembles the architecture of the control hierarchy such that every module is identified with a particular function in the hierarchy. Each module was realized separately in C and after debugging and testing, all the modules were compiled and linked together to build a single executable file, MACHINE.EXE. Figure 9.19 illustrates the functions assigned to different software modules and their layout in relation to the data flow between them. The functional description of each system module is given below.

**Sensor (input) module:** Reading of the input ports of the data acquisition board, recording of the position responses of the servomotors, and capturing of images of processed fish are carried out in this module. In order to reduce the effect of noise and ripple, all analog-to-digital-conversion (A/D) ports are read many times in rapid succession and the average is taken. All data are then converted into proper engineering units and transmitted to the pre-processor module.

**Preprocessor module:** This module contains the software required to interpret the sensor readings. Specifically, it transcribes crisp, sensory information into a linguistic form as required by the fuzzy knowledge base, in a compatible form.

**Knowledge base:** This module contains the rule-set and the fuzzy relational matrix derived from it. Since the relational matrix could be computed off line, the module simply stores the matrix in the form of a multidimensional array with each dimension corresponding to either an antecedent or a consequent variable. The size of this array is determined by the resolution of each fuzzy variable (the number of fuzzy states) and the total number of such variables.

**Decisional module:** Application of the compositional rule of inference is carried out in this module. The center of area method is applied to obtain a crisp value and the output is sent to the actuator (output) module.

**Actuator (output) module:** The crisp values of the consequent variables are scaled before writing them into the output ports of the digital-to-analog converter (D/A) within the PCL-812 board of the machine such that the analog signals produced could be directly interfaced with the corresponding servo amplifiers. The passing of tuning commands to the DMC-600 controller is also handled in this module.

The overall control of variable passing between modules and sequencing of various actions are coordinated by a main module, which incorporates all the modules into a single program. At run time, the user can monitor the machine through a menu-driven user interface, which facilitates interactive monitoring of each condition variable of the machine. Also, data may be stored for off line analysis and further evaluation.

#### 9.3.5.2 User interface of the machine

In the present implementation, the complexity of operation and the large quantity of information available for monitoring necessitate that the machine employ a user-friendly human-machine interface. The hierarchical structure of the control system itself provides a good basis for a hierarchically layered, menu-driven, user interface. The *Main Menu*, which is the top level of the user interface, provides a graphical representation that is analogous to the hierarchy of the machine. At any level of the interface hierarchy, the user can select and open up an item at a lower level or in the level of the hierarchy in order to monitor or interrogate a particular level or a device in the machine. A detailed description of each level of this interface will be given next.

Since the fish processing machine is intended for operation in an industrial process-plant environment, where it is used by ordinary, semi-skilled machine operators rather than computer experts, it is imperative that the software and user interface exhibit a great degree of user-friendliness and also robustness against abuse by users. The robustness of system software can be further enhanced after carrying out a large number of trial runs in an actual

plant environment. A menu-driven user interface provides a shield against abuse since users can select only a pre-determined set of actions. The user interface of the machine also disables many unused keys from the user console in order to reduce the risk of inadvertent inputs to critical system software.

The machine provides a fully-fledged data logging facility so that the user is able to recall and examine operating information of a previous run to investigate in detail the system performance. This feature enables the machine to save automatically the measurements from all feedback sensors including images from the CCD cameras. The user can enable or disable the data logging facility at will and also set the path and the file names for saved data.

### 9.3.6 Performance testing

Now, performance of the machine is studied using several categories of testing. Inferences obtained and the actions taken by the supervisory control system for different cases of machine context are given. Specifically, the performance of the knowledge-based decision-maker for various conditions of sensory data, such as servomotor responses of cutter and platform, cutter load profiles, and visual data from the inspection (secondary) camera, is discussed.

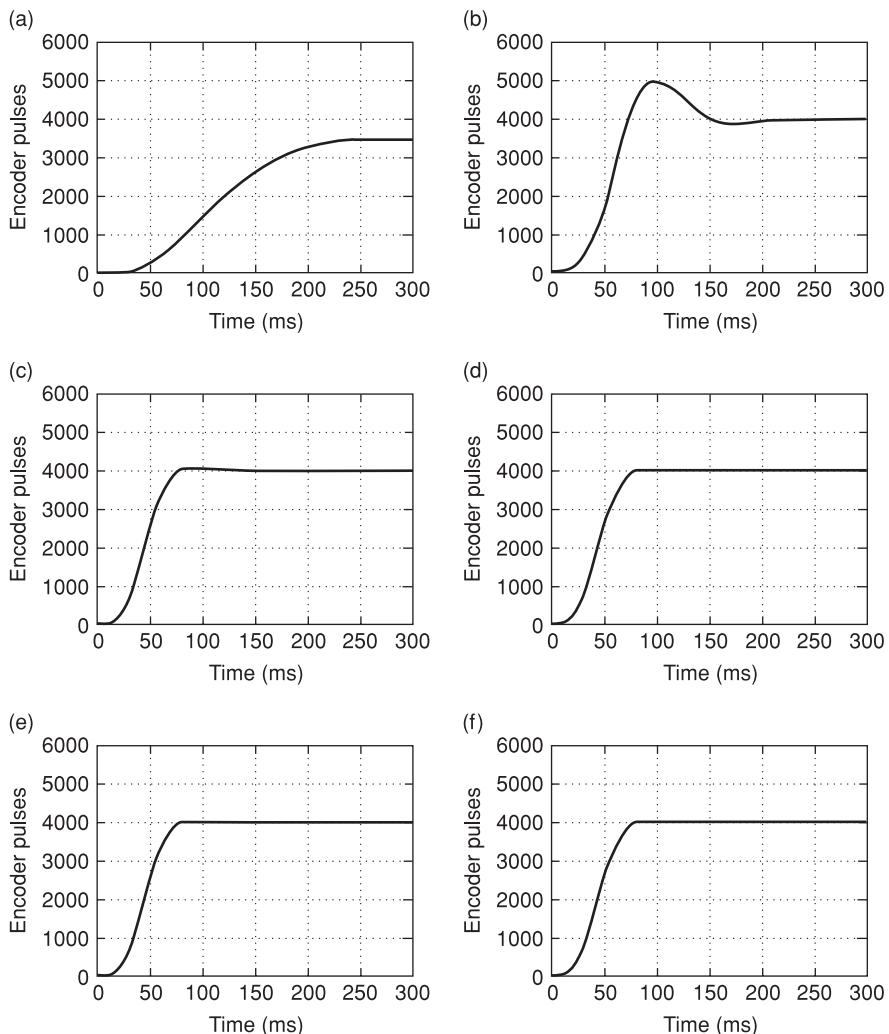
#### 9.3.6.1 Servomotor tuning examples

Several test runs were carried out on the cutter servomotor in order to investigate the behavior of the rule base for servomotor tuning. A series of step inputs of magnitude 4000 encoder pulses was applied to the cutter servomotor, and the response was preprocessed to obtain the context database. Then, using this information as the context for the rule base, a new set of parameters for the lead compensator and integrator was inferred. These parameters were then converted into their digital equivalent and the corresponding parameters of the servomotor controller (DMC-620) were updated to reflect the new values. This procedure was repeated for each test input.

**Initially overdamped system:** Figure 9.20(a) through (f) show six consecutive responses of the cutter servomotor for a series of step inputs of magnitude 4000 encoder pulses, while the on line tuning procedure is active. For this test, the initial controller parameters were chosen such that the cutter servomotor exhibited an overdamped response in the beginning. Table 9.10 shows the context, as generated by the preprocessor of the servomotor response. The columns (a) through (f) of Table 9.10 list the context generated for the responses in Figure 9.20(a) through (f), respectively.

Table 9.11 shows the controller parameters inferred by the knowledge-based servotuner for the response curves shown in Figure 9.20. The values listed are in the Z-domain.

**Initially underdamped system:** Figure 9.21(a) through (f) show six consecutive responses of the cutter servomotor for a series of step inputs of magnitude 4000 encoder pulses. For this test, the initial controller parameters were



**Figure 9.20:** Performance of the servomotor tuner for an initially overdamped system

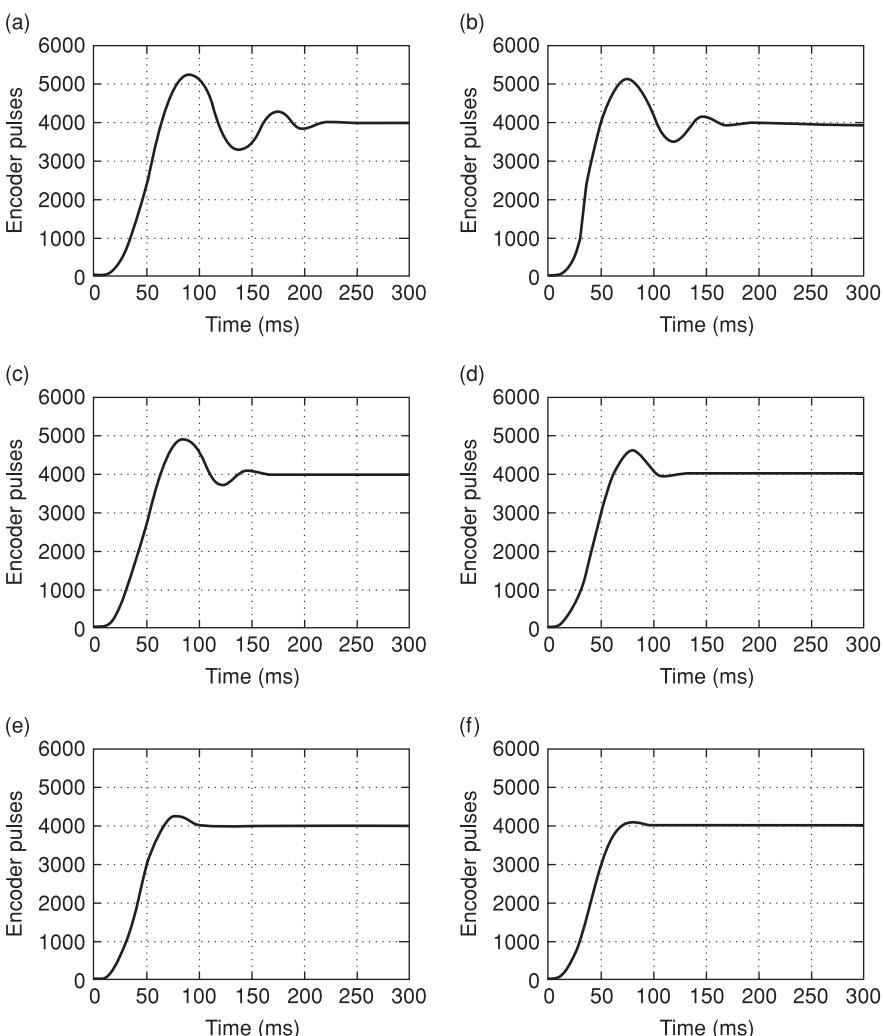
**Table 9.10:** Context corresponding to the servomotor responses in Figure 9.20

Performance parameter	(a)	(b)	(c)	(d)	(e)	(f)
Rise time	UN	IS	IS	IS	IS	IS
Damping ratio	UN	MD	IS	OS	IS	IS
Damped natural frequency	UN	MD	IS	IS	OS	OS
Overshoot	OS	UN	MD	OS	OS	OS
Offset	UN	MD	OS	OS	OS	OS

UN – Response is unsatisfactory, PR – Poor, MD – Moderate, IS – Within specification, OS – Over specification

**Table 9.11:** Inferences of the knowledge-based servomotor tuner for responses in Figure 9.20

Controller parameter	(a)	(b)	(c)	(d)	(e)	(f)	Z-domain value
Gain	0.413	1.666	8.678	13.605	18.482	19.551	
Pole	-0.7950	-0.5675	-0.7950	-0.8232	-0.8485	-0.8485	
Zero	0.9783	0.9483	0.9783	0.9830	0.9844	0.9844	

**Figure 9.21:** Performance of the servomotor tuner for an initially underdamped system

**Table 9.12:** Context corresponding to the servomotor responses in Figure 9.21

Performance parameter	Context database					
	(a)	(b)	(c)	(d)	(e)	(f)
Rise time	IS	OS	IS	OS	IS	IS
Damping ratio	PR	PR	MD	MD	IS	IS
Damped natural frequency	IS	MD	IS	IS	IS	IS
Overshoot	UN	UN	UN	UN	UN	IS
Offset	OS	OS	OS	OS	OS	OS

**Table 9.13:** Inferences of the knowledge-based servomotor tuner for responses in Figure 9.21

Controller parameter	Z-domain value					
	(a)	(b)	(c)	(d)	(e)	(f)
Gain	3.0939	4.3373	5.8941	8.2921	12.306	19.985
Pole	0.1298	0.0388	-0.0494	-0.1523	-0.2747	-0.4234
Zero	0.7750	0.8094	0.8380	0.8666	0.8954	0.9245

chosen such that the platform servomotor exhibited an underdamped response in the beginning. Table 9.12 gives the context as generated by the preprocessor corresponding to the servomotor response as shown in Figure 9.21. Table 9.13 gives the controller parameters (in the Z-domain) as inferred by the knowledge-based tuner, in this case.

The performance of the knowledge-based servomotor tuner, as illustrated in Figure 9.20 and Figure 9.21, appears to be quite satisfactory. The tuner has managed to bring the servomotor response to comply with the specifications, in approximately 5 to 6 cycles of tuning. For the purpose of these experiments, the initial controller parameters were set significantly off their normal operating values. In practice, however, perfect tuning is achieved much faster since the operating conditions usually remain close to their tuned performance.

The performance shown in Figure 9.20 and Figure 9.21 was obtained for the cutter servomotor (X-axis) that has been installed in the prototype machine. This servomotor is characterized by a high inertia load, arising due to the mass of the cutter assembly that is supported by the lead-screw, and a low level of damping. Also, there is a perturbation load that acts on the servomotor due to the flexible shafts of the cutter induction motors. This perturbation load varies in magnitude with the movement of the servomotor.

A knowledge-based system similar to the one for the X-axis carries out tuning of the delivery platform (Y-axis) servomotor. Experiments conducted on this servomotor have shown that the performance is very similar to that shown in Figure 9.20 and Figure 9.21. In this case, however, perfect tuning is achieved in a fewer number of cycles since the Y-axis servomotor carries a considerably smaller inertial load.

### 9.3.6.2 Machine tuning example

Performance of the machine tuner is illustrated now, using an example. First, several cutter load profiles were obtained through experimentation. Then, by the use of the cutter-load preprocessor, a context database, corresponding to each load profile, was generated. Next, the rule base for machine tuning was used to obtain the inferences pertaining to the conveyor speed and hold-down force.

Figure 9.22(a) through (d) show four cutter load profiles corresponding to actual fish cutting experiments carried out using the machine. Each curve

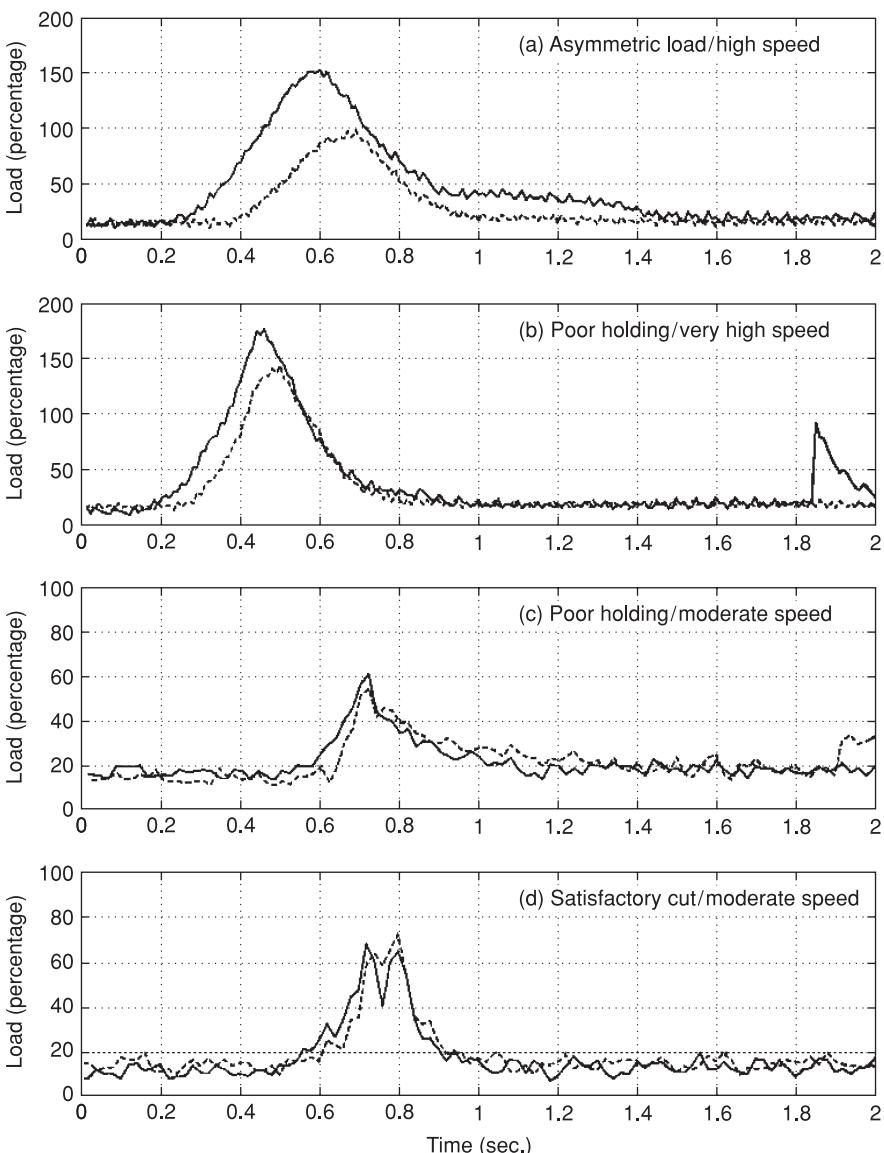
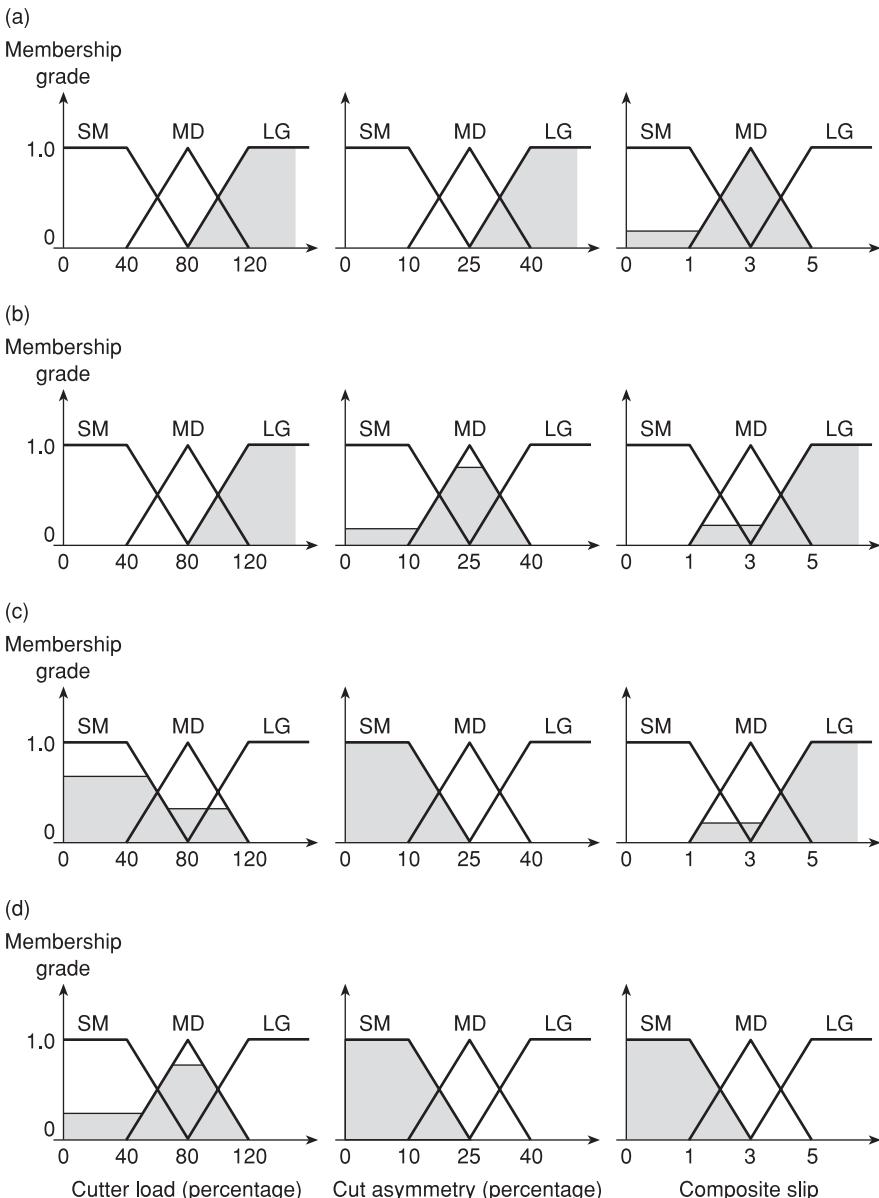


Figure 9.22: Cutter load profiles (solid curves: top blade; dashed curves: bottom blade)



**Figure 9.23:** Fuzzy linguistic context corresponding to the experimental cutter load profiles in Figure 9.22 (SM – small, MD – moderate, LG – large)

in Figure 9.22 has been obtained under different operating conditions of holding force and conveyor speed, as indicated. The numerical parameters resulting from preprocessing of these cutter load profiles are given in Table 9.14. The corresponding fuzzy linguistic context, as generated by the cutter-load preprocessor, is shown in Figure 9.23(a) through (d).

**Table 9.14:** Preprocessed cutter load information

Load Profile	Mean Load %	Asymmetry %	I <sup>st</sup> Peak (%) / Time (s) Top	I <sup>st</sup> Peak (%) / Time (s) Bottom	II <sup>nd</sup> Peak (%) / Time (s) Top	II <sup>nd</sup> Peak (%) / Time (s) Bottom
(a)	118	47	146/0.58	91/0.67	33/1.22	None
(b)	149	22	166/0.44	133/0.48	71/1.85	None
(c)	55	11	58/0.70	52/0.70	None	32/1.93
(d)	65	7.6	63/0.72	68/0.78	62/0.79	None

The rule base for machine tuning has two fuzzy inferences; namely, the change of conveyor speed and the magnitude of hold-down (grasping) force. Figure 9.24(a) through (d) illustrate the corresponding inferences when the knowledge-based system operates on the context shown in Figure 9.23. Figure 9.22(a) shows a case where the conveyor speed is quite high, as can be gauged from the peak load that is higher than normal. Also, in this case, it is clear that the fish has not been delivered symmetrically into the cutter, since there is a mismatch of peak loads corresponding to the top and the bottom load profiles. Furthermore, it is seen that the two peak values do not occur simultaneously because of the slight offset with an overlap, that has been provided in the placement of the two circular blades, so as to achieve a complete cut. The load profile of the top blade (solid curve) also shows a slight degree of slipping as can be seen from the higher than normal trailing region of the load profile. The fuzzy linguistic context obtained after preprocessing the load profile confirms those visual observations, as seen in Figure 9.23(a). The inferences corresponding to this context, shown in Figure 9.24(a), recommend a large reduction in conveyor speed and a moderate increase in hold-down force.

Figure 9.22(b) and (c) illustrate two cases where slipping occurs towards the tail end of the cutter load profiles. Here as well, the fuzzy linguistic context shown in Figure 9.23(b) and (c) captures the information as expected. Consequently, the knowledge-based system recommends tight holding and a reduced speed, as can be seen from Figure 9.24(b) and (c).

Figure 9.22(d) shows a satisfactory cut at a moderate conveyor speed. In this case, the two closely placed peaks visible on the load profile of the top blade are not due to slipping of fish, and the preprocessor correctly interprets this situation. The corresponding inferences suggest a soft holding force and a slight increase in conveyor speed to take into account the fact that the capacity of the cutter motors is underutilized in this case.

### 9.3.6.3 Product quality assessment example

An experimental result on product quality assessment by the machine monitoring system is given now. The knowledge-based monitor employs the information obtained from the secondary (inspection) CCD camera to derive an inference pertaining to the visual quality of the processed fish. Consider the visual data of Figure 9.25(a), which shows an image of a processed fish where

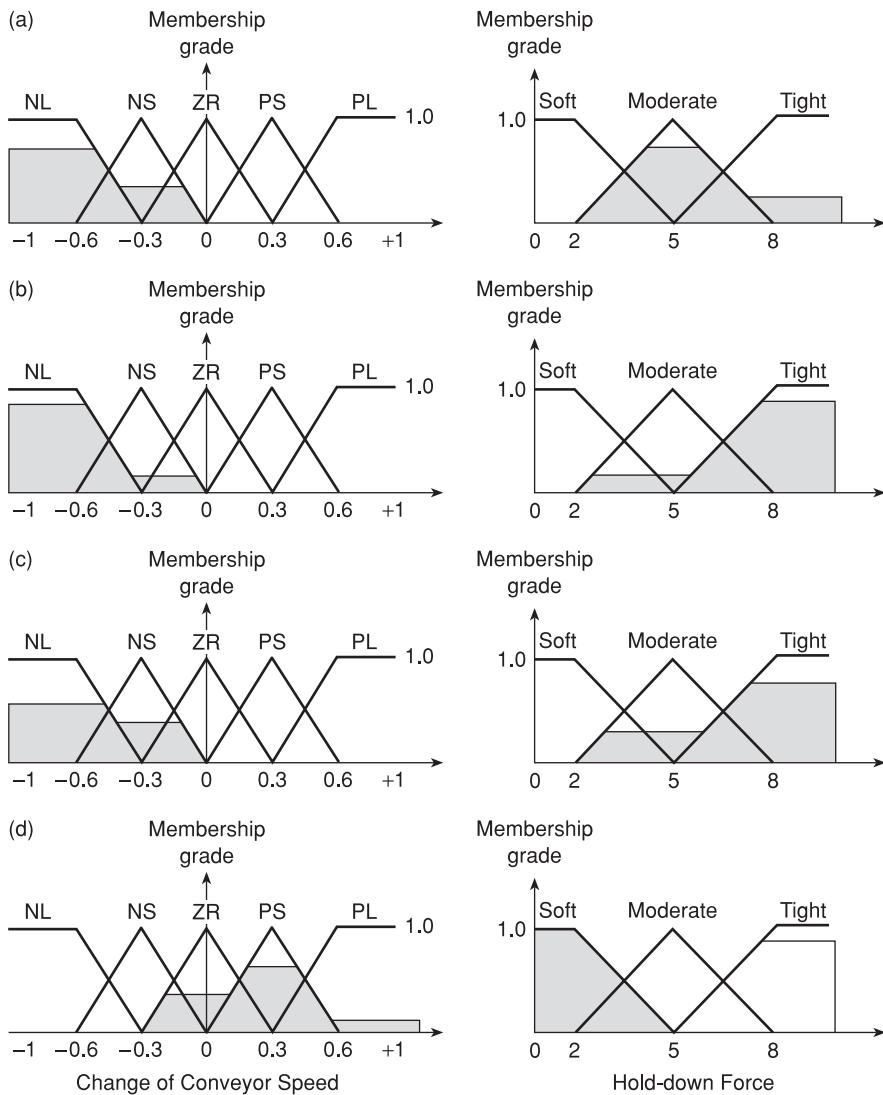
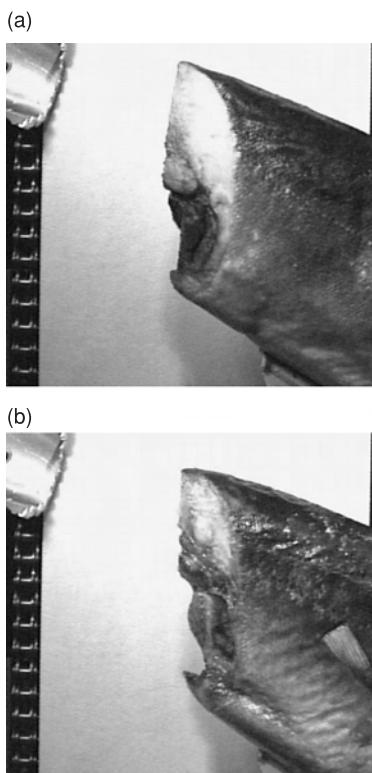


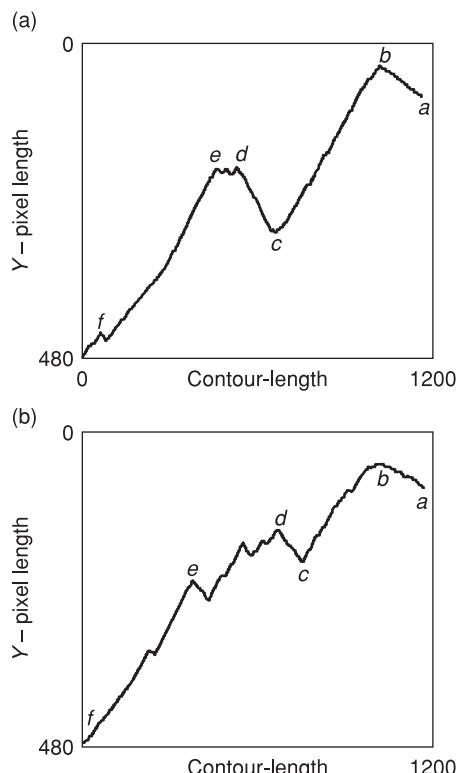
Figure 9.24: Fuzzy linguistic inferences of the machine tuner

the product quality is acceptable, and Figure 9.25(b) which illustrates a case where the product quality is not quite acceptable. Figure 9.25(a) and (b) show the arc-length profiles obtained through preliminary processing of the raw images. After preprocessing the arc-length profile of each image separately, the quality indicative indices listed in Table 9.15 and the corresponding fuzzy linguistic context shown in Figure 9.26 were obtained. The inference of the knowledge-based system for product quality assessment is a membership function expressed in terms of the three primary fuzzy sets of *Poor*,

### 9.3 Supervisory control of a fish processing machine



**Figure 9.25:** Camera images of processed fish: (a) Acceptable process quality. (b) Unacceptable process quality



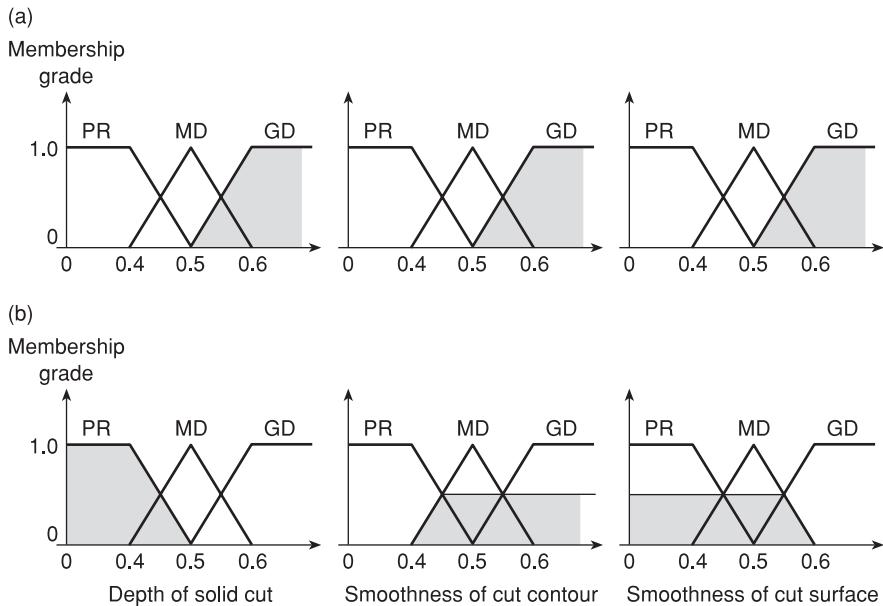
**Figure 9.26:** Arc-length profiles for processed fish: (a) Acceptable cut. (b) Unacceptable cut

**Table 9.15:** Quality indicative indices for the images of fish in Figure 9.25

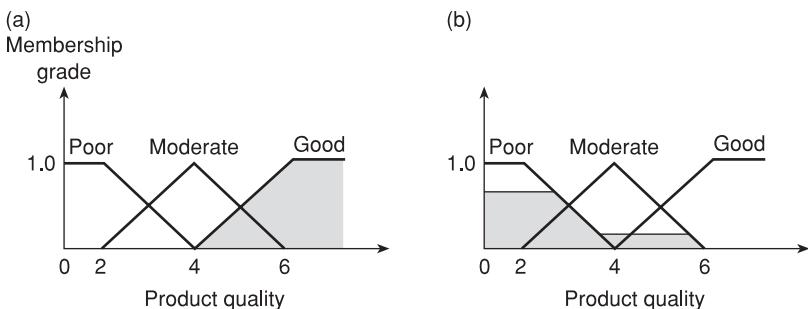
Parameter	Image (a)	Image (b)
Depth of solid section of cut	0.6387	0.3834
Smoothness of boundary contour of cut	0.6977	0.5501
Smoothness of cut surface	0.7300	0.4500

*Moderate*, and *Good*. The product quality, corresponding to the two images of fish in Figure 9.25, is shown in Figure 9.27(a) and (b).

Note that Figure 9.25(a) is an image of a processed fish with a clearly visible cut section. The depth and the width of the cut section are comparatively large and the cut surface is smooth. Also, the cut is readily discernible from the body of the fish by a smooth boundary contour. Consequently, this cut



**Figure 9.27:** Fuzzy linguistic context for the images of processed fish in Figure 9.25  
(PR – poor, MD – moderate, GD – good): (a) Good cut. (b) Poor cut



**Figure 9.28:** Fuzzy linguistic inference of product quality for images in Figure 9.25:  
(a) Good cut. (b) Poor cut

is assessed as of good quality. Preprocessing of the image in Figure 9.25(a) returns the fuzzy linguistic context shown in Figure 9.26(a), which clearly agrees with the visual observations. Running the knowledge base for the assessment of product quality over this context returns a fuzzy linguistic value of *Good*, as shown in Figure 9.27(a), which complies with the expected outcome.

Figure 9.25(b) illustrates a cut that has a comparatively small solid section of cut, with a poor depth. Also, the cutting region is not as smooth as in the previous case. As a result, this cut exemplifies a product of rather poor quality. The context corresponding to this image, shown in Figure 9.26(b),

confirms these observations. Consequently, the rule base returns a fuzzy linguistic value close to *Poor*, as shown in Figure 9.27(b), which is a reasonable outcome for this particular case.

## 9.4 Summary

Use of fuzzy logic and other techniques of soft computing in the design, development, and utilization of intelligent systems formed the theme of this book. This chapter concerned practical applications of fuzzy logic. Detailed case studies were presented, highlighting important practical issues of implementation and operation. The examples and case studies particularly concerned intelligent machines, intelligent control, knowledge-based tuning of controllers, hierarchical control architectures, intelligent supervisory control, and applications in the fish processing industry.

### Problems

**1.** Knowledge-based techniques are useful in injection molding of material. In particular, the following three situations may be considered:

- (i) Expert system to assist the development of molds.
- (ii) Intelligent controller for the injection molding process.
- (iii) Knowledge-based supervisor to monitor the process performance (e.g., defects in the molded parts) and to take corrective actions.

In (i) mold geometry, gate location, vent locations, etc., may be considered. In (ii) material characteristics, mixing, mold orientation, pressure, temperature, etc., are important factors. In (iii), problems such as poor density, hardness, and strength, and also nonuniformities and deformations in the molded parts are of importance. Discuss distinctive characteristics of the above three categories of application; and indicate whether fuzzy logic could be effectively used in each case.

**2.** High-speed, digital, fuzzy processor chips are commercially available with inference processing speeds of  $650 \mu\text{s}$  per cycle (with 20 rules). The function list of one such processor provides information on the following features:

- (a) Inference method
- (b) Defuzzification method
- (c) Number of I/O ports
- (d) Rule format
- (e) Number of rules
- (f) Rule weighting
- (g) Antecedent membership functions
- (h) Consequent membership functions.

Through an appropriate literature review, give the meaning of each of these terms.

**3.** What are advantages of an object-oriented implementation of a robotic workcell? Suppose that each machine tool of the workcell may be considered as a plant with

a set of actuators, sensors, and a controller. Sensors are monitored through polling, the information provided by them is interpreted (preprocessed), a control strategy is chosen, and the sensory information is supplied to the controller by the sensor manager. Discuss the use of fuzzy logic in controlling a workcell of this type.

4. Motion control in a constrained direction may result in very large and damaging forces. Similarly, force control in a free (unconstrained) direction can give rise to very large accelerations and decelerations which may lead to unstable behavior. Hybrid force-motion control, where force control along one axis and motion control along an orthogonal axis are implemented, may lead to stability, robustness, and task quality problems for these reasons. Could such problems be avoided by using an intelligent control approach? Compare it with impedance control where a relation between motion (velocity) and force is specified, and the controller attempts to reach that relation. A typical application would be in robotics; for example, a robotic hand carrying out a machining operation.
5. Consider vision-based robotics where CCD cameras and associated frame grabber/processor hardware/software, with various levels of image processing, are employed. Three situations are considered:
  - (a) A high-resolution camera with fast image processing for geometric and motion gauging, which information is then used in low-level control of the robot and other process components.
  - (b) A low-resolution camera is used to obtain static images of the work environment, which are then processed off line for planning the tasks; particularly, trajectory planning of the robot.
  - (c) A high-resolution camera is used on line for determining the “quality” of performance of the robotic system; for example, by analyzing the quality of the processed objects. Here the images are preprocessed to determine the necessary “features” of quality which in turn may be used in a rule-based system to make operational decisions on the robotic system.

Compare the three cases of robotic vision, identifying the distinguishing characteristics of each case.

6. A massaging robot has a moderately firm and elastic rotary wheel as the end-effector. The rotating speed  $\omega$  of the motor that drives the massaging wheel is not constant and changes with the load torque  $\tau$ . The normal force  $F$ , along the axis of rotation of the wheel, is adjustable. The trajectory of massage (both path and associated speed) is planned according to the needs and characteristics of the customer. Note that due to rubbing friction (assume a Coulomb friction model) the load torque  $\tau$  depends on  $F$ . Also, due to viscoelastic-type dissipation,  $\tau$  also depends on the rotary speed  $\omega$  and trajectory speed  $v$ .

One way to adjust  $F$  during massaging would be to use an impedance control technique, where a function that relates  $F$  to, say, the trajectory speed is specified as the required performance that is sought by the controller. Alternatively, a knowledge-based control approach may be employed. Outline such a technique and indicate the key considerations that are involved.

7. Robotic tasks may involve combinations of gross motion and fine manipulation. For example, in robotic microsurgery, moving a surgical appliance from point A to

point B may involve straightforward gross motions and coarse manipulation. But, actual performance of microsurgery will necessitate delicate and dexterous, fine manipulation. A variable-resolution controller might be suitable in such applications. Specifically, coarse-resolution controls will be used for gross manipulation, and fine resolution controls for micro-manipulation. Could fuzzy logic control be exploited in such a robotic application?

- 8.** How is the control of a flexible manufacturing (production) system (or facility control) different from that of a workcell or a machine tool?

Discuss the main considerations in the development of an intelligent control system for a workcell.

- 9.** Give reasons for the difficulty in implementing a customized control scheme in an industrial robot or machine tool.

Why is it difficult to build a workcell using machine tools, robots, etc. from different manufacturers?

- 10.** Consider a fuzzy logic-based self-tuning controller for a DC servo system. It is suggested that, instead of using this approach, which employs a fuzzy rule base, first a crisp *tuning curve* should be established, which uses a performance attribute of the control system as the independent variable. Then the tuning action may be established as a crisp decision, during operation of the system, simply by first computing the performance attribute using the system response, and then reading the tuning action from the pre-determined curve. Comment on this suggestion, emphasizing its general feasibility.

- 11.** Application of expert systems has been proposed for ship design. But, the development of the knowledge base is not straightforward. Knowledge may be acquired from published literature, experimental observation, and by interrogating domain experts. Then, one will have to address the “coupled” nature of the knowledge base. In particular, address the following questions:

- (a) How could one determine which cause affects which performance and by how much? For example, motion sickness may result from various factors such as the frequency and amplitude of motion, type and quantity of food eaten and how long ago, and the age and level of health of the person.
- (b) Once an uncoupled rule base is generated, how could one assign priorities or weights for the rules, and resolve conflicts?

- 12.** One may argue that it is not quite correct to talk about an “intelligent sensor.” Instead, one may speak of an “accurate” sensor, which faithfully reproduces the measurand (the signal that is being measured). It is what one does with the sensed data that could make the device intelligent. In this sense, it is more appropriate to treat a transducer, or a preprocessor, or a prefilter, or an interpreter as an intelligent device. Discuss this issue and redefine an intelligent sensor so as to remove this inconsistency.

- 13.** A specific sensor may provide several different abstractions of information corresponding to different layers of a hierarchical control system. Give an example of a sensor of this type and briefly describe its use.

- 14.** Consider a hierarchical architecture of a flexible manufacturing system and only the real-time tasks. A hierarchical control structure can be developed where

the supervisory (monitoring) level occupies the very top. The lowest level may constitute a set of services that are independent. What are typical functions of the supervisory level in this control structure?

15. Consider an automated process of grading (i.e., quality assessment) of a food product. (As a parallel problem, consider how the “beauty” of a person is determined by a human on the basis of various features.) Sensory means are used to obtain information that will establish such features as shape, size, weight, color, texture, and firmness. A rule-based system is used, with these features as inputs, to infer a “grade” (or level of quality) for each object. Comment on the fuzziness and information resolution of various levels of information in this process; particularly that of:
  - (a) Raw sensory information.
  - (b) Preprocessed information on object features.
  - (c) Knowledge-based decisions concerning product grade or quality.
16. Recent research shows that learning words and learning grammar are two quite different activities handled by different parts of the human brain. Can we consider this as a hierarchical problem where learning words is done at a lower level of the hierarchy, needing less intelligence than learning grammar?
17. Figure P9.17 shows a schematic representation of a knowledge-based system for quality assessment (grading) of processed herring roe (skeins). It uses pre-processed sensory information on shape, size, color, and texture, as obtained

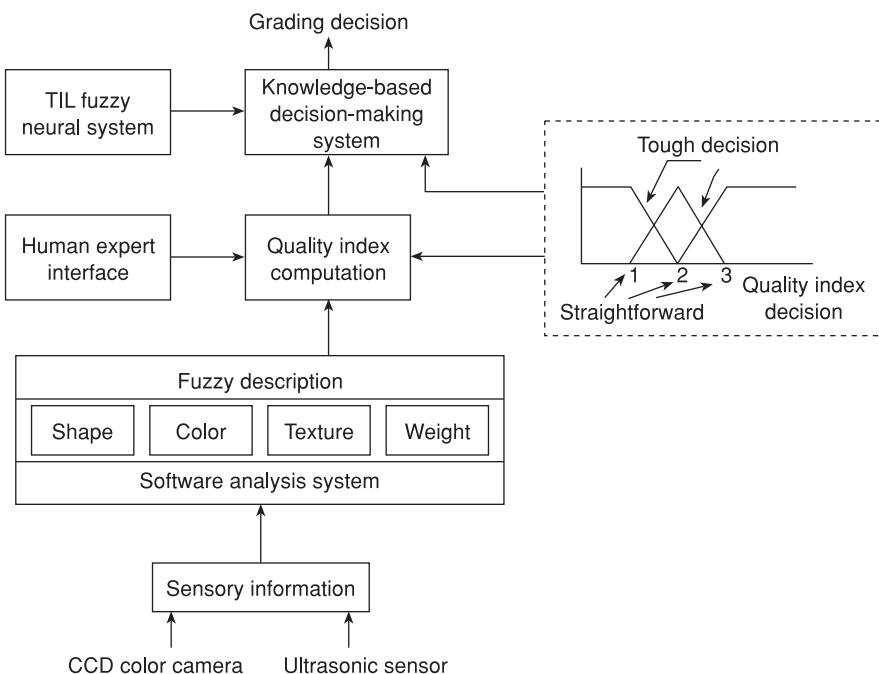
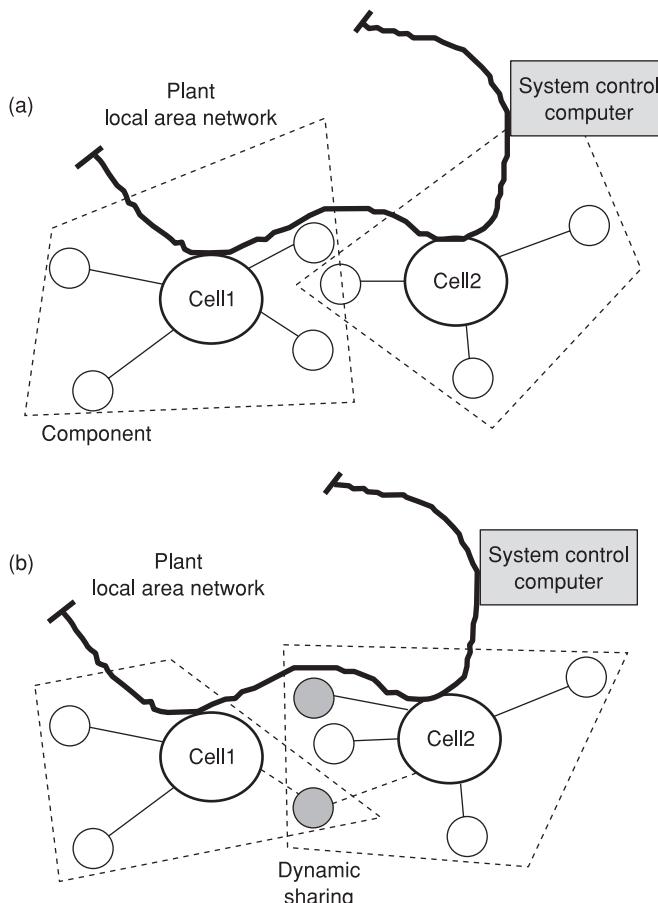


Figure P9.17: A knowledge-based system for grading herring roe

- from CCD cameras, ultrasound probes, etc. Expert graders are available who will provide grading criteria, and will serve as domain experts. Outline the process of developing a knowledge-based system for this application. Give a simplified rule base that can be implemented using fuzzy logic in this problem.
18. What are the advantages and drawbacks of distributed intelligence over central intelligence, in the context of a multi-component production system? In particular, do groups of objects perform more intelligently than individual objects with very little interaction?
19. Consider a supervisory intelligent system of a flexible manufacturing system. In particular, consider the *Task-Sequence Planner*, which is a subsystem of the supervisory system. Its tasks include assignment of various subtasks to active agents, coordination of the operation of the agents, supervision of the operation of these agents, and rescheduling in case of overloading, degradation, or malfunction. Indicate various needs of information/knowledge processing within this subsystem, and state whether fuzzy logic is suitable in each such need.
20. Hierarchical architecture, blackboard architecture, and object-oriented architecture are three possibilities for implementing the *knowledge system* of an intelligent controller. In fact, *blackboards* or *frames* and *objects* may be organized into hierarchical architectures and hence the latter two architectures are not mutually exclusive from the first. In a three-level hierarchy, the top layer may carry out non-real-time activities such as planning and scheduling while the intermediate layer may perform coordination or time-synchronization, and the bottom layer may perform low-level process executions in real time. Outline the advantages of a hierarchical structure of this type in comparison with a nonhierarchical blackboard or object-oriented implementation.
21. Sensors may be part of a workcell component, and may be termed “component sensors.” They may be used for various purposes such as low-level control and high-level monitoring. Alternatively, sensors may be considered in the context of monitoring workcell (or component) tasks. Here, the term “task sensor” is used. Identify the differences between component sensors and task sensors.
22. Development of a flexible production system will involve three hierarchical stages:
- (1) Sensor and controller integration for various components and machine tools (Lowest level).
  - (2) Integration of each workcell using the components in Stage 1. Inter-component communication that is compatible with an open architecture, so that non-proprietary components may be added, would be a consideration here (Intermediate level).
  - (3) Plant (facility) automation. Here, linking various workcells and developing the upper-level management systems will be involved (Top level).
- Flexibility is desirable, particularly when production in small batches (e.g., parts-on-demand) is needed. Note that reconfiguration and retooling of a non-flexible system can be quite costly and time consuming. Discuss how a flexible production system may cope with an unfamiliar or unexpected condition or disturbance input.
23. Figure P9.23 shows schematic representations of two stages of a flexible production system. Identify the category of the system and indicate when and why such a system would be advantageous. How could this system benefit from a knowledge-based (particularly fuzzy logic) approach?



**Figure P9.23:** Two stages of component configuration of a flexible production system

## References

1. De Silva, C.W., and MacFarlane, A.G.J. (1989) "Knowledge-based control approach for robotic manipulators," *International Journal of Control*, vol. 50, no. 1, pp. 249–73.
2. De Silva, C.W., and MacFarlane, A.G.J. (1989) *Knowledge-Based Control with Application to Robots*, Springer-Verlag, Berlin, Germany.
3. De Silva, C.W. (1991) "An analytical framework for knowledge-based tuning of servo controllers," *An International Journal, Engineering Applications of Artificial Intelligence*, vol. 4, no. 3, pp. 177–89.
4. De Silva, C.W., and Barlev, S. (1992) "Hardware implementation and evaluation of a knowledge-based tuner for a servo system," *Proc. IFAC Symposium ACASP*, Grenoble, France, pp. 293–8.
5. De Silva, C.W. (1992) "Research laboratory for fish processing automation," *Int. J. Robotics and Computer-Integrated Manufacturing*, vol. 9(1), pp. 49–60.

6. De Silva, C.W., Gosine, R.G., Wu, Q.M., Wickramarachchi, N., and Beatty, A. (1993) "Flexible automation of fish processing," *International Journal of Engineering Applications of Artificial Intelligence*, vol. 6, no. 2, pp. 165–78.
7. De Silva, C.W. (1993) "Soft automation of industrial processes," *International Journal of Engineering Applications of Artificial Intelligence*, vol. 6, no. 2, pp. 87–90.
8. De Silva, C.W., and Gu, J. (1994) "An intelligent system for dynamic sharing of workcell components in process automation," *International Journal of Engineering Applications of Artificial Intelligence*, vol. 7, no. 5, pp. 571–86.
9. De Silva, C.W., and Lee, T.H. (1994) "Fuzzy logic in process control," *Measurements and Control*, vol. 28, no. 3, pp. 114–24, June.
10. De Silva, C.W. (1995) *Intelligent Control: Fuzzy Logic Applications*: CRC Press, Boca Raton, FL.
11. De Silva, C.W., Gamage, L.B., and Gosine, R.G. (1995) "An intelligent firmness sensor for an automated herring roe grader," *International Journal of Intelligent Automation and Soft Computing*, vol. 1, no. 1, pp. 99–114.
12. De Silva, C.W. (1995) "Applications of fuzzy logic in the control of robotic manipulators," *Fuzzy Sets and Systems*, vol. 70, no. 2–3, pp. 223–4.
13. De Silva, C.W. (1997) "Intelligent control of robotic systems," *International Journal of Robotics and Autonomous Systems*, vol. 21, pp. 221–37.
14. De Silva, C.W., and Wickramarachchi, N. (1998) "Knowledge-based supervisory control system of a fish processing workcell; part I: system development," *Engineering Applications of Artificial Intelligence – Int. J. Intelligent Real-Time Automation*, vol. 11, no. 1, pp. 97–118, May.
15. De Silva, C.W., and Wickramarachchi, N. (1998) "Knowledge-based supervisory control system of a fish processing workcell; part II: implementation and evaluation," *Engineering Applications of Artificial Intelligence – Int. J. Intelligent Real-Time Automation*, vol. 11, no. 1, pp. 119–34, May.
16. De Silva, C.W. (2003) "The role of soft computing in intelligent machines," *Philosophical Transactions of the Royal Society*, Series A, UK (In Press).
17. Goulet, J.F., de Silva, C.W., Modi, V.J., and Misra, A.K. (2001) "Hierarchical control of a space-based deployable manipulator using fuzzy logic," *AIAA Journal of Guidance, Control, and Dynamics*, vol. 24, no. 2, pp. 395–405.
18. Hu, B.G., Gosine, R.G., Cao, L.X., and de Silva, C.W. (1998) "Application of a fuzzy classification technique in computer grading of fish products," *IEEE Transactions on Fuzzy Systems*, vol. 6, no. 1, pp. 144–52, February.
19. Lee, M.F.R., de Silva, C.W., Croft, E.A., and Wu, Q.M.J. (2000) "Machine vision system for curved surface inspection," *Machine Vision and Applications*, vol. 12, pp. 177–88.
20. Modi, V.J., Zhang, J., and de Silva, C.W. (2003) "Intelligent hierarchical modal control of a novel manipulator with slewing and deployable links," *Acta Astronautica* (In Press).
21. Rahbari, R., and de Silva, C.W. (2001) "Comparison of two inference methods for P-type fuzzy logic control through experimental investigation using a hydraulic manipulator," *International Journal Engineering Applications of Artificial Intelligence*, vol. 14, no. 6, pp. 763–84.
22. Tang, P.L., Poo, A.N., and de Silva, C.W. (2001) "Knowledge-based extension of model-referenced adaptive control with application to an industrial process," *Journal of Intelligent and Fuzzy Systems*, vol. 10, no. 3, 4, pp. 159–83.
23. Wu, Q.M., and de Silva, C.W. (1996) "Dynamic switching of fuzzy resolution in knowledge-based self-tuning control," *Journal of Intelligent and Fuzzy Systems*, vol. 4, no. 1, pp. 75–87.

24. Yan, G.K.C., de Silva, C.W., and Wang, G.X. (2001) "Experimental modeling and intelligent control of a wood-drying kiln," *International Journal of Adaptive Control and Signal Processing*, vol. 15, pp. 787–814.
25. Yasunobu, S., Miyamoto, S., and Ihara, H. (1983) "Fuzzy control for automatic train operation systems," *Proc. 4th IFAC/IFIP/IFORS International Congress on Control in Transportation Systems*, Baden-Baden.
26. Ziegler, J.G., and Nichols, N.B. (1942) "Optimum settings for automatic controllers," *Trans. ASME*, 64, 759–68.

# Tools of soft computing in real-world applications

In this chapter we present several case studies showing the power of applying techniques of soft computing for dealing with complex real-world applications pertaining to such fields as control systems design, stabilization of high-order power systems, communication systems, layout planning for resources management, and civil engineering applications.

Some of these case studies have been published in the open literature by the co-authors and/or their students. This chapter is not meant to be exhaustive in terms of the case studies involved, but is intended to assist those in the field in tackling complex systems using the tools developed throughout the textbook.

The first case study tackles the problem of intelligent tuning of a class of controllers using fuzzy logic systems, neural networks, and genetic algorithms. The second case study tackles the aspect of layout planning and how to approach it from a soft computing perspective. The third case study outlines the different steps for dealing with stability and controller design for a power plant system. The fourth case study deals with a communication network-based problem pertaining to mobile position estimation using an RBF network in CDMA cellular systems. The fifth case study tackles resource optimization in ATM networks using machine learning and soft computing techniques.

## Case study 1

### Expert parameter tuning of DC motor controller

## 1 Introduction

We present in this case study results pertaining to the on line adaptation mechanism of a class of linear controllers (taken here as PID-based controllers). PID controllers are usually simple to implement and reliable controllers which perform well after being tuned appropriately to certain plants. However, they do not adapt well when circumstances are changed due to control signal disturbances or wide variations of applied loads. In this case study, three different control schemes based on soft computing techniques are used for enhancing the performance of a conventional PID controller of a DC motor. Fuzzy-based methods with and without parameter optimization are implemented. These are compared with the results obtained through implementation of a dynamic neural network. It is shown that significant improvements can be made over the conventional static PID controller when using fuzzy and neural methods, in particular for load disturbance recovery.

## 2 Background

The proportional, integral, and derivative (PID) controllers have been widely used in various industrial control processes. In fact one estimate [6] suggests that PID controllers are controlling some 90% of industrial processes. The reason for such popular use of PID controllers can be attributed to their simple structure and robust performance in a wide range of operating conditions. The design of such controllers requires specification of three parameters, namely proportional gain, integral gain, and derivative gain. The important issue also known as the tuning problem in PID controller design is the appropriate setting of these three gains. The conventional approach to determining the PID parameters is to study the mathematical models of the process and to try to come up with a simple tuning law that will provide a fixed set of gain parameters. One well-known example of such an approach is the Ziegler–Nichols method [22].

It is well known that PID controllers are adequate enough for the processes that could be modeled using linear first or second order systems. However, real industrial processes may have characteristics such as high-order, dead-time, nonlinearity, etc. which make the modeling of these processes with simple linear systems occasionally inaccurate. In addition, these processes may also be affected by parameter variations caused by aging components,

noise, and load disturbances. The tuning laws provided by the model-based approaches are no longer sufficient to set the controller gains properly for these complex processes. In the last couple of decades, there have been some efforts to improve the tuning laws obtained using model-based approaches and reported in the literature [3, 4].

These difficulties in finding PID gain tuning laws for the complex process led to the development of some on line parameter adjustment methods for self-tuning PID controllers [9, 5]. The idea behind these conventional self-tuning PID control schemes is to tune the PID gain at any instance based on a structurally fixed parameter-evolving process model produced by an on line identification procedure. There are two problems commonly reported in the literature associated with these conventional self-tuning methods. First, the computational demand required by the on line identification of the process model and the momentary tuning of the PID gains appears to be very high. Second, since a simple *a priori* model structure is assumed for the identification procedure, structural disturbance such as load disturbance on the process is not handled well by these conventional self-tuning schemes. Robust control techniques have been proposed to tackle some of these issues. While successful in particular applications, these techniques have led to higher order controllers and an increased burden on system computational resources.

In recent years, there has been growing interest in designing soft computing-based self-tuning PID control schemes. Soft computing tools such as those based on fuzzy logic, neural networks, and genetic algorithms offer knowledge representation and learning capabilities to tackle a wide range of complex dynamical systems, which may be ill-defined or subjected to largely varying parameters. Using these tools, different categories of control methods have been investigated [15, 14]. Auto-tuning of PID controllers has been studied extensively using fuzzy logic and neural networks. A fuzzy gain scheduling scheme of PID controllers based on the error and its first difference is reported in [21]. In [16], a fuzzy self-tuning of PID controllers is presented by parameterizing the Ziegler–Nichols tuning formula by a single parameter which is determined using a Mamdani type of FIS. A fuzzy logic-based gain tuner for self-tuning PID gains is proposed in [8] in conjunction with a Lyapunov function-based stabilizer. A neural network-based auto-tuner in which the neural network outputs the PID gains is reported in the literature. In [18], the network is trained using Ziegler–Nichols' method for determining target PID gains. In [1], a method is described for open-loop or closed-loop auto-tuning in which optimal PID gains are computed by training a neural network initialized with Ziegler–Nichols values. A neuromorphic self-tuning PID control scheme is proposed in [2] which implements a closed-loop system with backpropagation training using output error signals.

In this case study, we expand on some of the tools developed by others and integrate them with appropriate algorithms to come up with a more efficient means for approaching a class of dynamical system taken here as a DC motor operating under widely varying operating conditions and load disturbances.

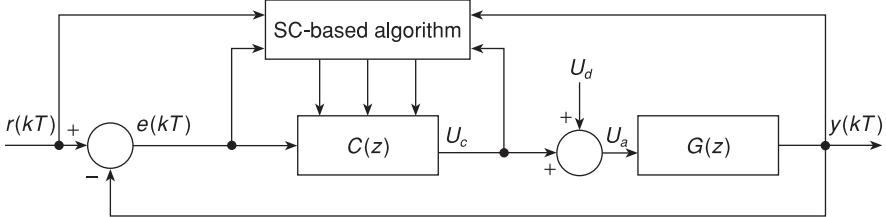


Figure 10.1: Soft computing-based self-tuning PID control scheme

### 3 Design problem statement

The main goal in this work is to design systematic approaches aiming at adjusting PID parameters on line so that the closed-loop system behaves in a stable manner with desired performance characteristics. This objective is achieved by designing a scheme using soft computing techniques such as fuzzy logic and neural networks for tuning the parameters of a PID controller for achieving constant motor speed under several load conditions. As shown in Figure 10.1, the idea behind the self-tuning PID control of the DC motor is to design an algorithm using soft computing techniques for updating controller gains dynamically based on some input variables. These input variables are selected from the set of variables which include current and previous values of output (motor speed), control input (applied voltage), reference (motor speed reference), error signal (reference – output).

The DC motor model is considered in discrete-time representation as follows:

$$\begin{aligned} y(kT) = & a_1 y((k-1)T) + \dots + a_n y((k-n)T) \\ & + b_1 u((k-1)T) + \dots + b_m u((k-m)T) \end{aligned} \quad (1)$$

where  $T$  is the sampling period,  $y(kT)$  is the motor speed and  $u(kT)$  is the applied voltage at  $k$ -th sample. The input voltage to the motor consists of the controller output  $u_c(kT)$  and the input disturbance  $u_d(kT)$ . The PID controller has the following discrete form:

$$\begin{aligned} u_c(kT) = & u_c((k-1)T) + K_P(e(kT) - e((k-1)T)) + K_I e(kT) \\ & + K_D(e(kT) - 2e((k-1)T) + e((k-2)T)) \end{aligned} \quad (2)$$

where  $e(kT) = Ref(kT) - y(kT)$ , and  $K_P$ ,  $K_I$ , and  $K_D$  represent, respectively, the proportional, integral, and derivative gain parameters of the PID controller which should be adjusted using the self-tuning scheme.

### 4 DC motor model description

In order to illustrate the capabilities of the soft computing-based self-tuning PID control algorithms, simulations are carried out on a DC motor model.

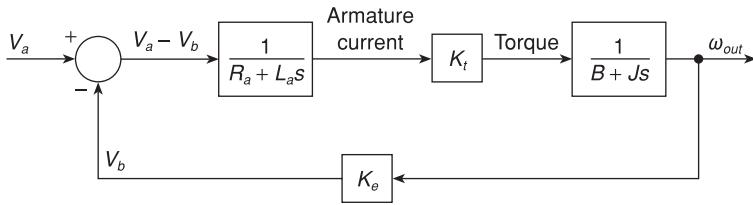


Figure 10.2: Transfer function block diagram of DC motor system

Table 10.1: DC motor parameter values

Motor parameters	Values
Armature Resistance ( $R_a$ )	3.5 Ω
Armature Inductance ( $L_a$ )	10.0 mH
Back emf ( $K_e$ )	1.1 V/rad/sec
Torque Constant ( $K_t$ )	1.088 Nm/A
Total Inertia ( $J$ )	0.0945 kg-m

The transfer function of the motor is derived from the data given in [11]. The values of the motor model parameters as shown in Figure 10.2 are listed in Table 10.1. Using these parameter values, the following system time constants are determined:

$$T_e = \frac{L_a}{R_a} = 0.0029 \text{ sec} \quad (3)$$

$$T_m = \frac{R_a J}{K_e K_t} = 0.28 \text{ sec}$$

Since  $L_a \ll \frac{R_a^2 J}{K_e K_t}$ , the transfer function of the motor is simplified to

$$G(s) = \frac{y(s)}{u(s)} = \frac{\left(\frac{1}{K_e}\right)\left(\frac{60}{2\pi}\right)}{(1 + T_e s)(1 + T_m s)} \left(\frac{rpm}{V}\right) = \frac{K}{s^2 + 2\zeta\omega_n + \omega_n^2} \left(\frac{rpm}{V}\right) \quad (4)$$

where  $K = 10994$ ,  $\zeta = 4.97$ , and  $\omega_n = 35.59$ . To simulate the motor model, this transfer function is discretized using the zero order hold (ZOH) and a sampling period of 0.05 sec. The discrete-time model is given in the form of equation (1) as follows:

$$y(kT) = a_1 y((k-1)T) + b_1 u((k-1)T) + b_2 u((k-2)T) \quad (5)$$

where  $a_1 = 0.8345$ ,  $b_1 = 1.3611$ , and  $b_2 = 0.0757$ .

**Table 10.2:** Model parameters for varying total inertia

Model type	$J$	$K$	$\omega_n$	$\zeta$	$a_1$	$b_1$	$b_2$
model_Jnom	0.0945	10994	35.59	4.97	0.8345	1.3611	0.0757
model_Jlow	0.0756	13743	39.79	4.46	0.7976	1.6665	0.0907
model_Jhigh	0.1417	73295	29.06	6.06	0.8864	0.9330	0.0534

## 5 Conventional PID tuning

To evaluate the results obtained using soft computing-based PID self-tuning schemes, a conventional tuning formula [13] is used to derive a fixed set of PID gains. In [13], the internal model control formula is given for deriving PID gains for the second-order model of equation (4) as follows:

$$\begin{aligned} K_p &= \frac{2\zeta\omega_n}{\lambda K} \\ K_I &= \left( \frac{K_p\omega_n}{2\zeta} \right) T \\ K_D &= \left( \frac{K_p}{2\zeta\omega_n} \right) \left( \frac{1}{T} \right) \end{aligned} \quad (6)$$

where  $T$  is the sampling period 0.05 sec and  $\lambda$  is a user specified parameter that corresponds to the closed-loop speed of response. With appropriately selected  $\lambda$  value for some desired performance, the PID gains are obtained as  $K_p = 0.6$ ,  $K_I = 0.1$ , and  $K_D = 0.05$ . Also to evaluate the robustness of the proposed schemes, three different models are considered by varying the total load inertia of DC motor model as shown in Table 10.2.

## 6 Fuzzy logic-based self-tuning

In this section, a fuzzy logic-based approach for self-tuning of PID gains is described (Figure 10.3). A self-tuning scheme is developed using the Takagi–Sugeno (TS) fuzzy inference systems (FIS). In [20], it has been shown that using TS FIS, one can construct nonlinear variable gain controllers. This means that using direct fuzzy control based on the TS FIS for motor speed control is equivalent to using PID controller with on line tuning of its gains. In other words, the PID controller with self-tuning capability can be realized using direct Sugeno FIS control with error ( $e(n)$ ), delta-error ( $de(n)$ ), and delta-delta-error ( $dde(n)$ ) as the inputs to the fuzzy controller and the rule consequent is chosen as the PID control in incremental form:

$$\Delta u(n) = K_p de(n) + K_I e(n) + K_D dde(n) \quad (7)$$

where  $e(n) = r(n) - y(n)$ ,  $de(n) = e(n) - e(n-1)$ , and  $dde(n) = de(n) - de(n-1)$ .

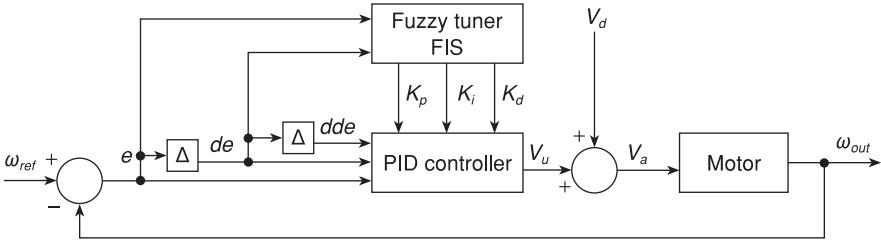


Figure 10.3: Fuzzy logic-based self-tuning PID control scheme

However, in [20], the above result is proved by assuming a simplified TS control rule scheme. In this scheme, all the rule consequents use a common function and are proportional to one another. While reducing the number of adjustable parameters needed in the consequent part of the rules, this scheme seems to restrict the variability of gains independent of each other. This implies that all the gains share the same nonlinear characteristics. It may be advantageous to consider the possibility of the situation where some gains are made to vary independently of the rest of the nonlinear gains which share the same nonlinear characteristics. With this partial constraint on the rule consequents, a proof similar to the one in [20] can be obtained showing that a TS control scheme with partial proportionality constraint is equivalent to the nonlinear variable PID control.

Consider the  $j$ -th TS fuzzy rule with partially constrained rule consequent in incremental PID control form:

$$\begin{aligned} R_j : & \text{If } e(n) \text{ is } E_j \text{ AND } de(n) \text{ is } DE_j \text{ AND } dde(n) \text{ is } DDE_j \\ & \text{Then } \Delta u(n) = k_{j1} K_I e(n) + k_{j2} (K_P de(n) + K_D dde(n)) \end{aligned} \quad (8)$$

where  $k_{j1} \neq k_{j2}$  in general, and  $k_{11} = k_{12} = 1$  without loss of generality. Now, the overall control action produced by this TS control scheme is given as

$$\begin{aligned} \Delta u(n) &= \frac{\sum_{j=1}^{\Omega} \mu_j \Delta u_j(n)}{\sum_{j=1}^{\Omega} \mu_j} \\ &= \frac{\sum_{j=1}^{\Omega} \mu_j (k_{j1} K_I e(n) + k_{j2} (K_P de(n) + K_D dde(n)))}{\sum_{j=1}^{\Omega} \mu_j} \\ &= \frac{\sum_{j=1}^{\Omega} \mu_j k_{j1}}{\sum_{j=1}^{\Omega} \mu_j} K_I e(n) + \frac{\sum_{j=1}^{\Omega} \mu_j k_{j2}}{\sum_{j=1}^{\Omega} \mu_j} (K_P de(n) + K_D dde(n)) \\ &= G_1(\vec{k}_1, \vec{\mu}, \vec{x}) K_I e(n) + G_2(\vec{k}_2, \vec{\mu}, \vec{x}) (K_P de(n) + K_D dde(n)) \end{aligned} \quad (9)$$

where  $\Omega$  is the total number of rules,  $\mu$  is the membership functions used, and  $\vec{x}$  is the vector input variables, namely  $e(n)$ ,  $de(n)$ , and  $dde(n)$ . The nonlinear variable gains  $G_1$  and  $G_2$  are given as

$$\begin{aligned} G_1 &= \frac{\sum_{j=1}^{\Omega} \mu_j k_{j1}}{\sum_{j=1}^{\Omega} \mu_j} \\ G_2 &= \frac{\sum_{j=1}^{\Omega} \mu_j k_{j2}}{\sum_{j=1}^{\Omega} \mu_j} \end{aligned} \quad (10)$$

It can be seen from the above derivation that direct TS fuzzy control becomes nonlinear gain PID control and when  $k_{j1} = k_{j2}$ , the above proof and the expressions for nonlinear gains become identical to those given in [20]. Having the proportionality constants  $k_{j1}$  and  $k_{j2}$  decoupled is beneficial for intelligent motor control, especially when dealing with set point and load disturbances.

## 7 Fuzzy logic-based self-tuning with GA optimizer

In this section, the methodology to optimize the membership function parameters of the TS FIS using a genetic algorithm is described. First, the GA with real-value encoding is outlined briefly describing the genetic operators used. Then the fitness function used in the GA optimization to quantify the performance of the fuzzy self-tuning scheme is presented.

### 7.1 Genetic algorithms

Genetic algorithms (described in detail in Chapter 8) are search algorithms based on the operations observed in natural selection and genetics [10]. Generally, the search variables optimized by a GA are encoded into binary strings called chromosomes. The genetic algorithm deals with a population of chromosomes, each one of them representing a possible solution for a given problem. Each chromosome is associated with a fitness value that indicates the *goodness* of its proposed solution.

In general, the use of a genetic algorithm requires the implementation of six fundamental steps: chromosome representation, the creation of an initial population, the genetic operators making up the reproduction function, selection function, termination criterion, and the evaluation function. The steps are described in the rest of this section.

A chromosome representation is needed to describe each individual in the population of solutions used in the GA. The representation scheme determines how the problem is structured in the GA and also determines the genetic operators that are used. It has been shown that more natural representations are more efficient and produce better solutions in a GA optimization [12]. In this case study, a simple GA variant with a chromosome representation of floating-point values is used to optimize the membership function parameters of the fuzzy self-tuning scheme. In this application, the chromosome represents the membership function parameters and the

associated fitness value is a quantitative measure of the self-tuning controller performance. For this GA optimization, the initial population is generated randomly with uniformly distributed parameter values within their bounds.

Genetic operators provide the basic search mechanism of the GA. There are two basic types of genetic operators: crossover and mutation. These operators are used to create new solutions based on the existing solutions in the population. Crossover takes two individuals and produces two new individuals while mutation alters one individual to produce a single new solution. For the type of chromosome representation used in this paper, the following two genetic operators are implemented in the reproduction step of the GA: arithmetic crossover operator and non-uniform mutation operator. For real parameter solution vectors  $X$  and  $Y$ , let  $a_i$  and  $b_i$  be the lower and upper bound, respectively, for each parameter  $i$ .

*Arithmetic crossover operator:* this crossover generates a random number  $r = U(0, 1)$  from a uniform distribution and creates two new individuals ( $X'$  and  $Y'$ ) by producing two complementary linear combinations of the parents ( $X$  and  $Y$ ).

$$\begin{aligned} X' &= rX + (1 - r)Y \\ Y' &= (1 - r)X + rY \end{aligned} \quad (11)$$

*Non-uniform mutation operator:* this mutation operator randomly selects one variable,  $x_i$ , and sets it equal to a non-uniform random number as follows:

$$x'_i = \begin{cases} x_i + (b_i - x_i)f(G) & \text{if } r_1 \leq 0.5 \text{ and } i = j \\ x_i - (a_i + x_i)f(G) & \text{if } r_1 \geq 0.5 \text{ and } i = j \\ x_i & \text{otherwise} \end{cases} \quad (12)$$

where

$$f(G) = \left( r^2 \left( 1 - \frac{G}{G_{max}} \right) \right)^b \quad (13)$$

and  $r_1 = U(0, 1)$ ,  $r_2 = U(0, 1)$  are uniform random numbers between  $[0, 1]$ ,  $G$  is the current generation,  $G_{max}$  is the maximum number of generations,  $b$  is a shape parameter.

The reproduction step explores the solution space through genetic operations by undergoing crossover and mutation. After reproduction, the fitness of each chromosome is evaluated using the fitness function which is described in the next section. The selection of individuals to produce successive generations plays an important role in a genetic algorithm. A probabilistic selection is performed based upon the individual's fitness such that the better individuals have an increased chance of being selected. There are several schemes for the selection process and roulette wheel selection is adopted for the GA implementation used in this case study.

The GA continues the reproduction and selection steps until a termination criterion is met. A specified maximum number of generations is used as a termination criterion for the GA used here.

## 7.2 The fitness function for self-tuning PID control

To evaluate the fitness of a chromosome, i.e., a set of membership function parameters, its associated TS fuzzy self-tuning of the PID controller must be simulated for the DC motor model. For this purpose, the membership function parameters from each chromosome of the GA are used in the simulation of the fuzzy self-tuning PID controller. From each of these simulation performances, the following three performance measures are computed to form the fitness function: normalized sum of absolute errors,  $e_{sum}$ , normalized overshoot,  $M_p$ , and the normalized settling time,  $t_s$ . These three performance measures for the output response of the DC motor with step and load disturbances are combined to form the following fitness function:

$$J = w_e(1 - e_{sum}) + w_m(1 - M_p) + w_t(1 - t_s) \quad (14)$$

where  $w_e$ ,  $w_m$ , and  $w_t$  are the weights to specify the relative importance of each individual objective function to the global fitness function which is maximized by the GA.

## 8 Neural network-based self-tuning

In this section, a self-tuning PID control scheme in which a neural network is used to tune the parameters of the controller is described in detail. The self-tuning PID control scheme using a neural network is illustrated in Figure 10.4. The neural network used in this scheme generates the PID parameters in an on line manner based on past input and/or output data history. This is

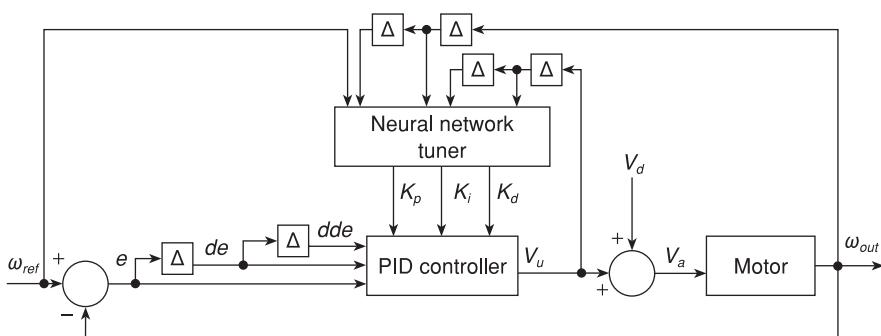


Figure 10.4: Soft computing-based self-tuning PID control scheme

similar to the case of conventional PID on line tuning in which a human operator adjusts the PID parameters based on the past data history of the system. In the following subsection, a *specialized learning* scheme [7] based on backpropagation learning is derived for training of the neural network-based self-tuning PID control scheme.

### 8.1 Backpropagation learning for neural network-based self-tuning PID control

As shown in Figure 10.4, the connection weights of the neural network are adjusted such that the error cost function is minimized.

$$E = \frac{1}{2}e^2(n+1) = \frac{1}{2}(r(n+1) - y(n+1))^2 \quad (15)$$

The neural network architecture considered here is a feedforward one with  $Q$  number of layers and  $m$  number of inputs. Since the outputs of the neural network are the PID gains  $K_P$ ,  $K_I$ , and  $K_D$ , the output layer of the network has three neurons. The activation functions of the hidden layer and output layer nodes are of the sigmoid type.

The basic idea here is how to use the output error  $e(n+1)$  as a learning signal for the multilayer feedforward neural network to adjust the weights and to minimize the error cost function  $E$ . In the standard backpropagation algorithm, the weights of the neural network can be updated by backpropagating the learning signal when the derivative  $\frac{\partial E}{\partial w}$  can be expanded using the chain rule. This chain rule can be extended to update the weights of the neural network which generates the PID gains. Based on the steepest descent method, the weight update rule at the output layer is given as

$$\Delta w_{kj}^Q(n+1) = -\eta \frac{\partial E}{\partial w_{kj}^Q} + \alpha \Delta w_{kj}^Q(n) \quad (16)$$

where  $\eta > 0$  and  $\alpha \geq 0$  are the learning rate and momentum term, respectively [19]. At the  $q$ -th hidden layer, the weight update rule is given as

$$\Delta w_{ji}^q(n+1) = -\eta \frac{\partial E}{\partial w_{ji}^q} + \alpha \Delta w_{ji}^q(n) \quad (17)$$

Let us define

$$\delta_k^Q = -\frac{\partial E}{\partial net_k^Q} \quad (18)$$

where  $net_k^Q = \sum_j w_{kj}^Q o_j + \theta_k^Q$ . Let the output of neuron  $k$  at the output layer be denoted by  $o_k$ ,  $k = 1, 2, 3$ . Then the PID gains are given as

$$\begin{aligned} K_P &= k_1 o_1 \\ K_I &= k_2 o_2 \\ K_D &= k_3 o_3 \end{aligned} \tag{19}$$

where  $k_1$ ,  $k_2$ , and  $k_3$  are the positive constant values which scale the neural network outputs to produce the PID gains.

The derivative  $\frac{\partial E}{\partial w}$  is calculated throughout the plant and currently used PID controller by the chain rule

$$\frac{\partial E}{\partial w_{kj}^Q} = \frac{\partial E}{\partial net_k^Q} \frac{\partial net_k^Q}{\partial w_{kj}^Q} = \frac{\partial E}{\partial net_k^Q} o_j^{Q-1} \tag{20}$$

and

$$\delta_k^Q = -\frac{\partial E}{\partial net_k^Q} = -\frac{\partial E}{\partial y(n+1)} \frac{\partial y(n+1)}{\partial u(n)} \frac{\partial u(n)}{\partial o_k} \frac{\partial o_k}{\partial net_k^Q} \tag{21}$$

where

$$\begin{aligned} \frac{\partial E}{\partial y(n+1)} &= \frac{\partial E}{\partial e(n+1)} \frac{\partial e(n+1)}{\partial y(n+1)} = -(r(n+1) - y(n+1)) = -e(n+1) \\ \frac{\partial o_k}{\partial net_k^Q} &= f'(net_k^Q) = o_k(1 - o_k) \end{aligned} \tag{22}$$

$$\frac{\partial u(n)}{\partial o_k} = \begin{cases} k_1(e(n) - e(n-1)) & \text{for } k = 1 \\ k_2 e(n) & \text{for } k = 2 \\ k_3(e(n) - 2e(n-1) + e(n-2)) & \text{for } k = 3 \end{cases}$$

Therefore

$$\delta_k^Q = e(n+1) \frac{\partial y(n+1)}{\partial u(n)} o_k(1 - o_k) \frac{\partial u(n)}{\partial o_k} \tag{23}$$

and the weight update rule at the output layer becomes

$$\Delta w_{kj}^Q(n+1) = \eta \delta_k^Q o_j^{Q-1} + \alpha \Delta w_{kj}^Q(n) \tag{24}$$

For the  $q$ -th hidden layer, we have

$$\begin{aligned} \frac{\partial E}{\partial net_j^q} &= \sum_k \frac{\partial E}{\partial net_k^{q+1}} \frac{\partial net_k^{q+1}}{\partial o_j^q} \frac{\partial o_j^q}{\partial net_j^q} \\ &= -\sum_k \delta_k^{q+1} w_{kj}^{q+1} o_j^q (1 - o_j^q) \end{aligned} \tag{25}$$

and

$$\delta_j^q = -\frac{\partial E}{\partial \text{net}_j^q} = -\sum_k \delta_k^{q+1} w_{kj}^{q+1} o_j^q (1 - o_j^q) \quad (26)$$

for  $q = Q - 1, \dots, 1$ . Thus the weight update rule for the  $q$ -th hidden layer becomes

$$\Delta w_{ji}^q(n+1) = \eta \delta_j^q o_i^{q-1} + \alpha \Delta w_{ji}^q(n) \quad (27)$$

To calculate  $\delta_k$ , the Jacobian  $\frac{\partial y(n+1)}{\partial u(n)}$  is needed and it can be estimated from the system dynamics equation. Here, the DC motor model is known and the Jacobian is readily computed as

$$\frac{\partial y(n+1)}{\partial u(n)} = b_1 \quad (28)$$

However, if the system is not known, then the Jacobian is estimated using the system emulator or some approximation method [17].

## 9 Implementation results and discussion

Sugeno FIS uses error and delta-error as inputs. The error input variable is a fuzzy set with 3 membership functions, representing NB, ZO, and PB. The other input variable delta-error is also a fuzzy set, but with two membership functions, representing big and small. The PID gains  $K_p$ ,  $K_i$ , and  $K_D$  are computed through a set of rules, and there are six rules used in the TS FIS. For each rule, there are three output consequents each being selected as constant values. According to Sugeno inferencing, the total output is computed using a weighted average of all the rules output, where each rule has its firing strength as weight.

For our implementation of the Sugeno FIS, we choose a FIS with the following settings: The logical AND is selected as the Min operator and for the implication, we use the Mamdani Min impicator. The aggregation operation is performed using the weighted average operator. The input membership functions are formed using equally spaced triangular and trapezoidal membership functions. The output function for each gain is simply a constant, as stated above. These constants were originally set equal to the optimal non-fuzzy controller gains for the nominal model, and then tuned manually to get better performance. We assume that the range of operations for the motor will be around 10 to 40 most of the time. We therefore set the ranges for the input functions as follows: range for error is  $[-20, 20]$  and for delta-error it is  $[-15, 15]$ .

In Figure 10.5, the motor speed response with Sugeno-based PID self-tuning is shown for step input and load disturbances and compared against

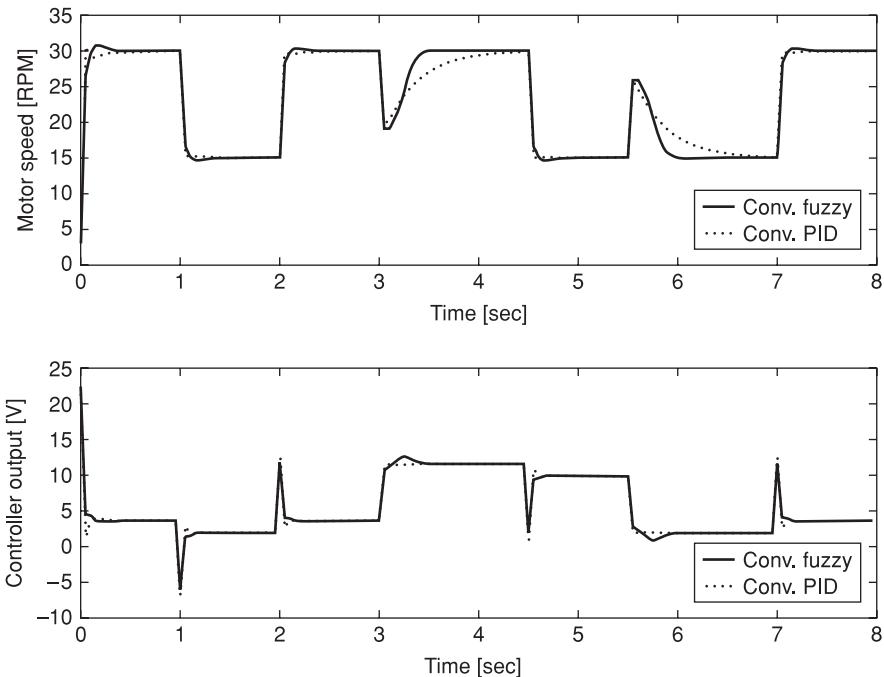


Figure 10.5: Comparison of conventional fuzzy self-tuning fixed gain PID

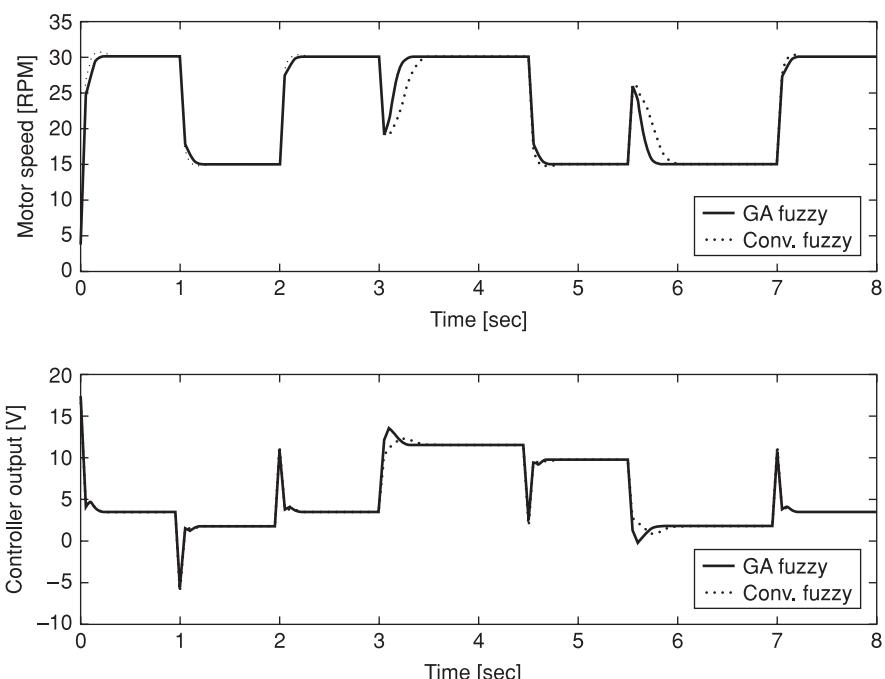
the response generated using conventionally tuned PID gains. The performance of the TS FIS-based self-tuning scheme for model parameter changes is shown in Figures 10.7, 10.9 and 10.11.

To verify the applicability of the neural network-based self-tuning PID control scheme, the simulations of the proposed scheme as shown in Figure 10.4 are carried out on the second-order discrete-time model of the DC motor. First, after some trial and error experiments, the architecture of the neural network is chosen as a three-layer feedforward network with 5, 20 and 3 neurons in the input, hidden, and output layers, respectively. The input variables include the previous two values for each of the output and control input and the current reference value. The three outputs of the neural network are scaled by constant values  $k_1 = 1.0$ ,  $k_2 = 0.5$ , and  $k_3 = 0.05$  as given in equation (15). The hidden layer and the output layer nodes use the sigmoid activation function.

The training of the network is performed using the backpropagation algorithm with specialized learning as described in section 2.4. The initial weights and biases are randomly chosen between  $-0.05$  and  $0.05$ . Because of these small initial weights, the initial PID gains produced by the neural network would be very close to 0.5 times the scaling factors. These initial PID gains are selected such that the initial neural network starts with some stable responses.

The input–output exemplar for the backpropagation training is obtained directly from the closed-loop simulation where the target value used is the reference input to the closed-loop system. The error between the reference input and the predicted output is backpropagated to adjust the neural network weights as described in section 2.4. The reference inputs to the closed-loop system are chosen as a sequence of pseudo-random step changes between 3 rpm and 30 rpm with each step input lasting for 10 simulation samples (0.5 sec). Also, an input disturbance of 5V is introduced to teach the neural network about different load conditions of the motor speed control. After the neural network has converged in the 500th trial, the motor speed response, the control signal, and the PID gains are depicted in Figure 10.13 for set point and load disturbances. The performance of the neural network-based self-tuning scheme for model parameter changes is shown in Figures 10.14, 10.15 and 10.16 for model  $J_{low}$  and model  $J_{high}$ .

It can be seen from Figures 10.5–10.16 that the self-tuning schemes of PID gains based on the soft computing tools perform significantly better for different loading conditions. That is, when a disturbance is applied at the input to the motor, the speed recovery is faster whenever the PID gains are adjusted on line. Also, it can be seen from Figures 10.5 and 10.13 that the TS FIS-based self-tuning scheme and the neural network-based self-tuning scheme perform well on the nominal motor model. These performances can be attributed to the learning capability of the neural network-based scheme



**Figure 10.6:** Comparison of conventional and GA optimized fuzzy self-tuning

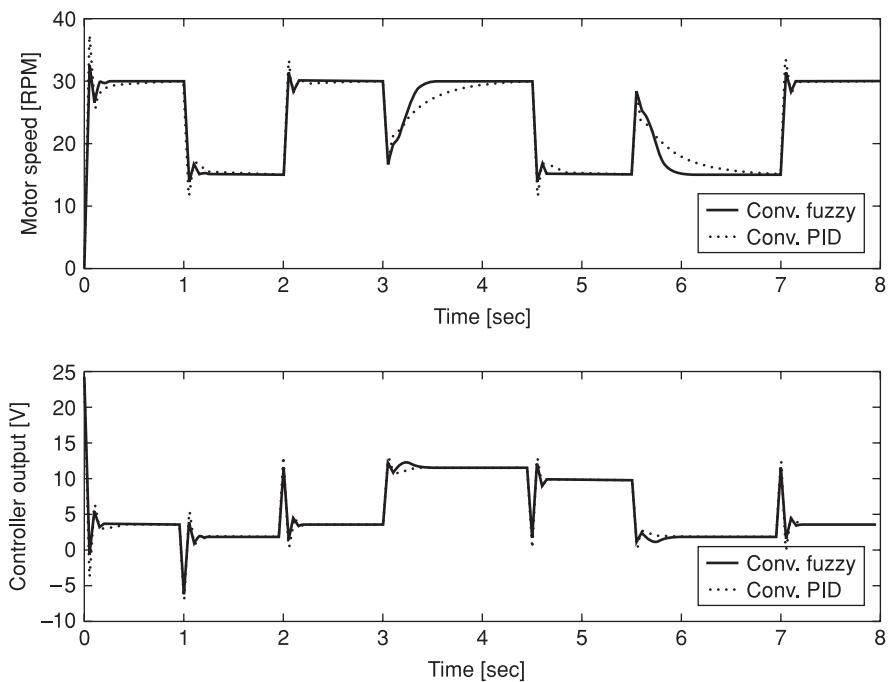


Figure 10.7: Comparison of conventional fuzzy self-tuning fixed gain PID for low  $J$

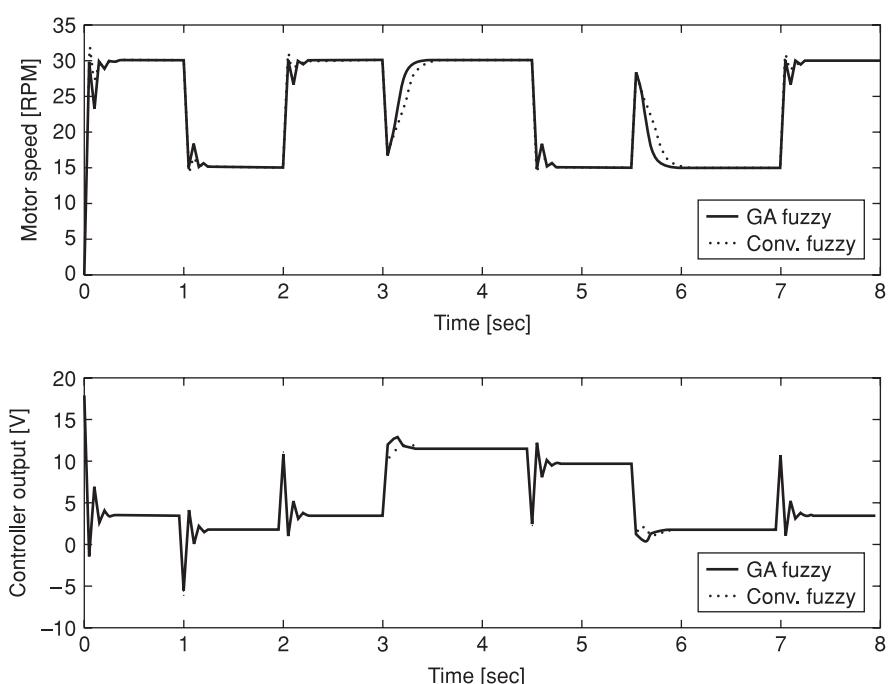


Figure 10.8: Comparison of conventional and GA optimized fuzzy self-tuning for low  $J$

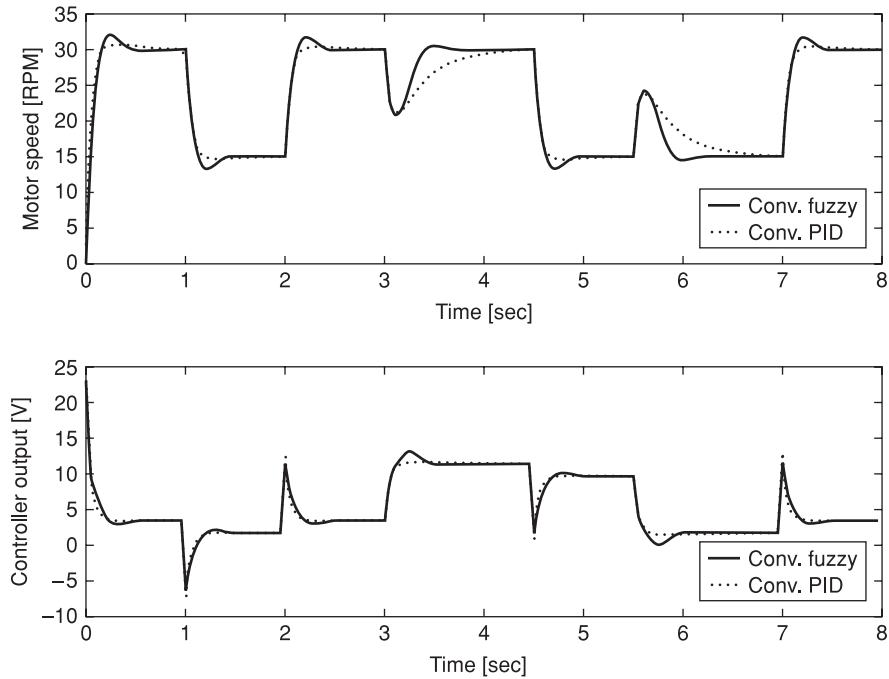


Figure 10.9: Comparison of conventional fuzzy self-tuning fixed gain PID for high  $J$

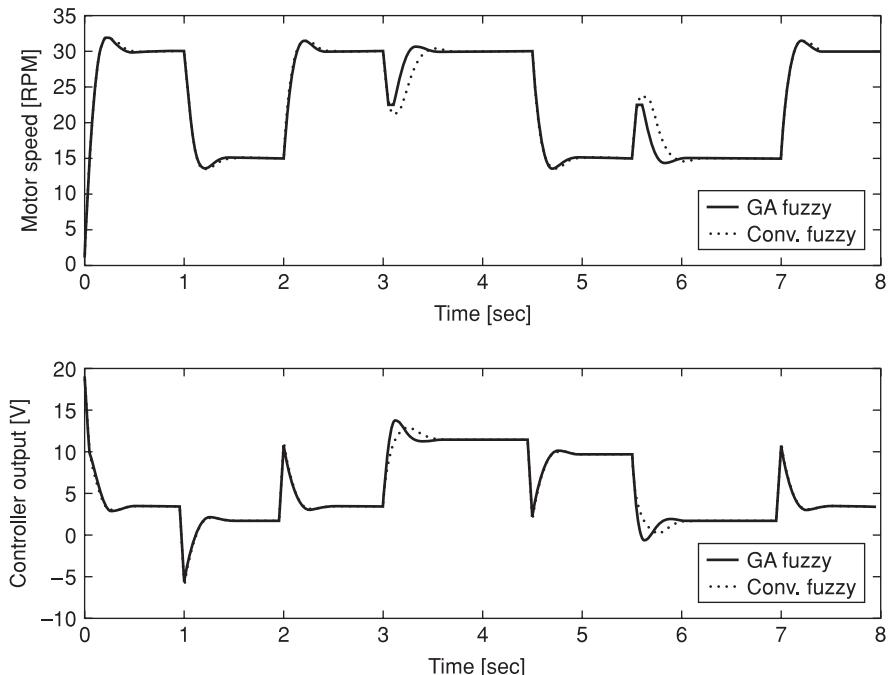


Figure 10.10: Comparison of conventional and GA optimized fuzzy self-tuning for high  $J$

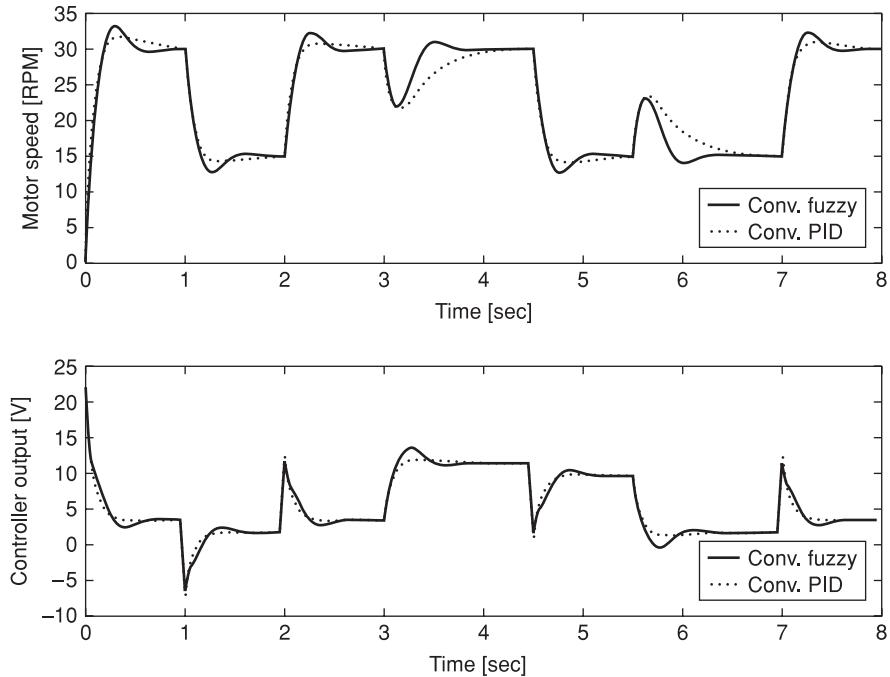


Figure 10.11: Comparison of conventional fuzzy self-tuning fixed gain PID for high  $J$

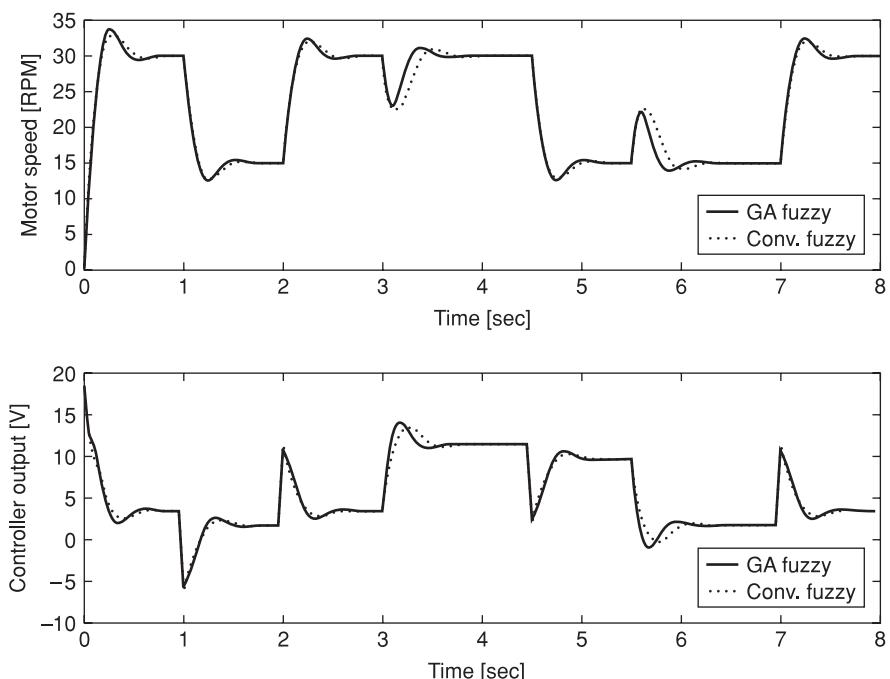


Figure 10.12: Comparison of conventional and GA optimized fuzzy self-tuning for high  $J$

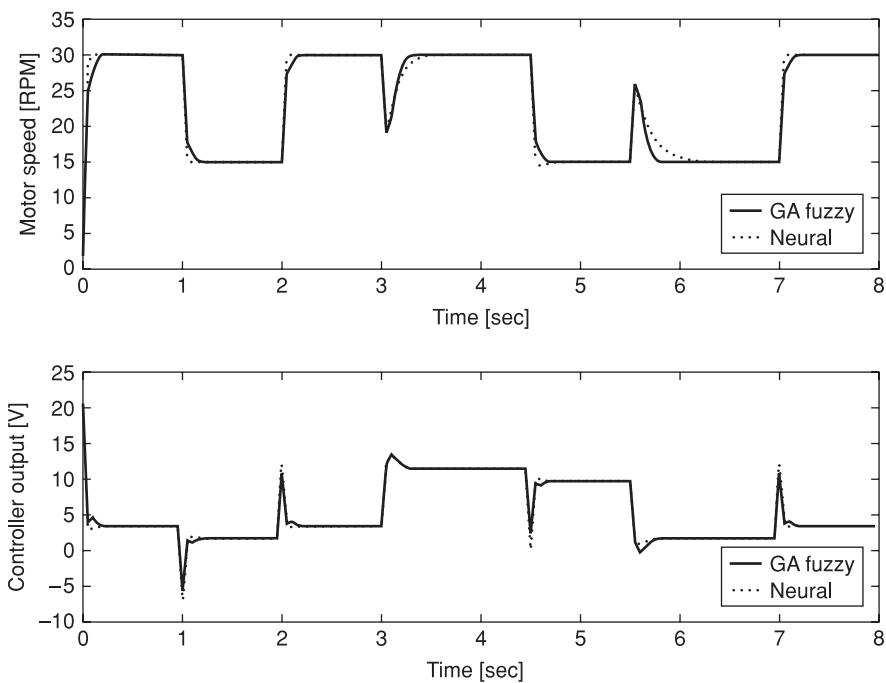


Figure 10.13: Comparison of GA optimized fuzzy and NN-based self-tuning

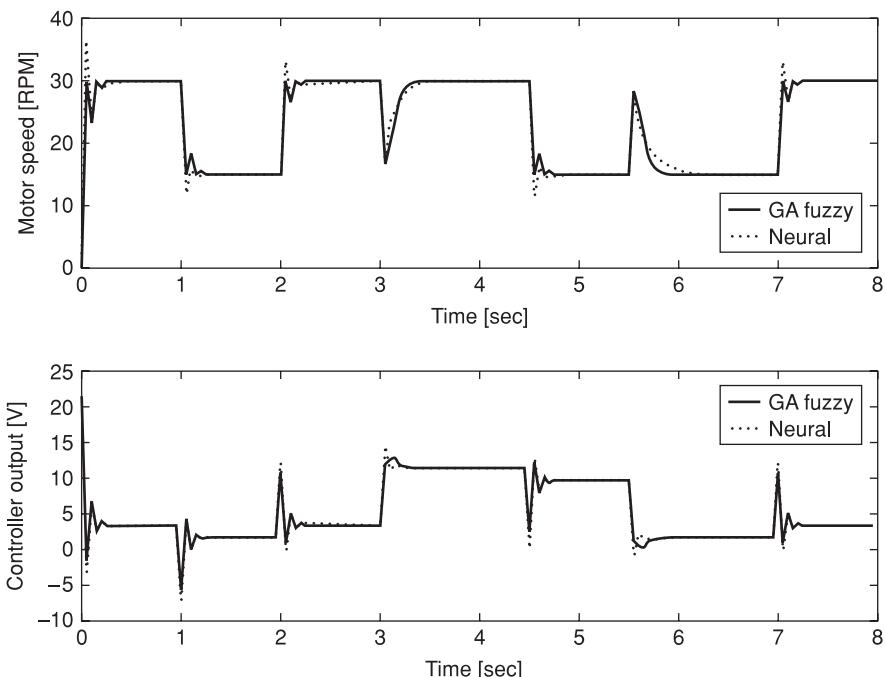


Figure 10.14: Comparison of GA optimized fuzzy and NN-based self-tuning for low  $J$

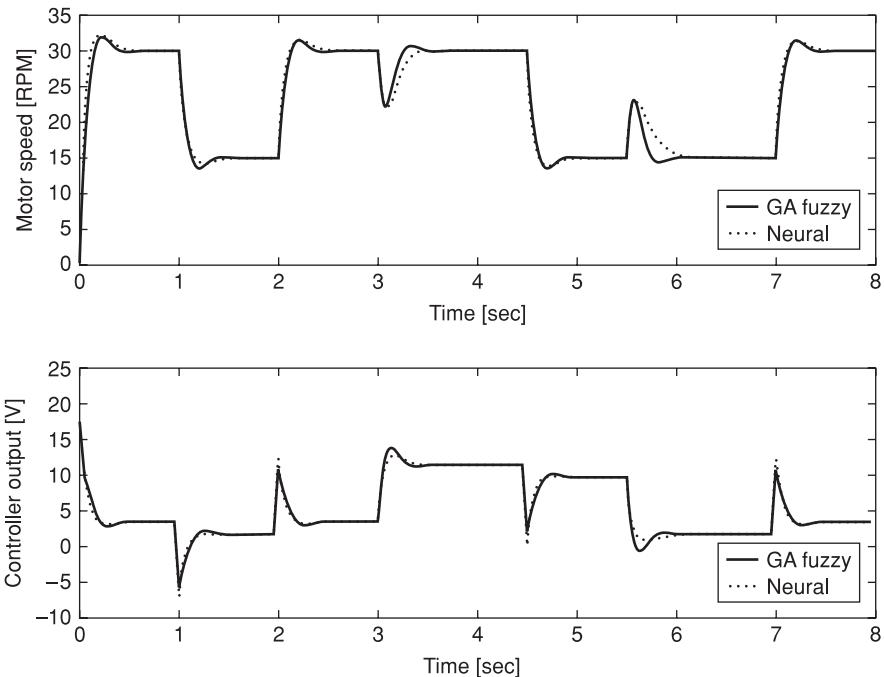


Figure 10.15: Comparison of GA optimized fuzzy and NN-based self-tuning for high  $J$

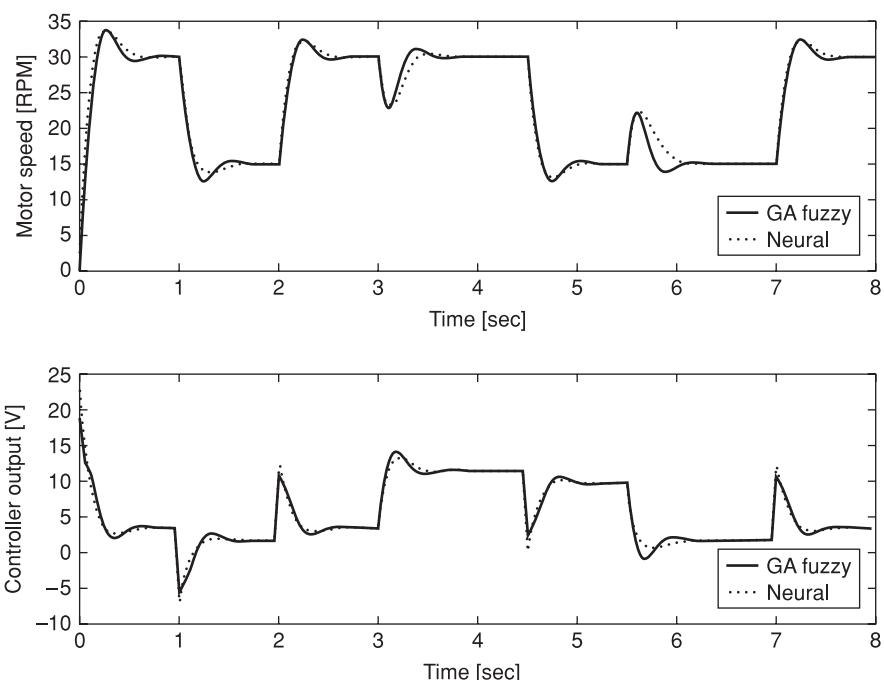


Figure 10.16: Comparison of GA optimized fuzzy and NN-based self-tuning for high  $J$

and the ease-of-tuning property of the modified TS fuzzy rule-based self-tuning scheme.

It can also be seen from Figures 10.5–10.16 that all the self-tuning schemes have some difficulties in coming up with better performance compared to fixed gain PID when the model parameter values change significantly. For the neural network-based self-tuning scheme, this can be explained using the fact that the training is performed only on the nominal model. So, it seems the neural network needs further training to come up with better generalization capability for different model dynamics. In the case of fuzzy logic-based self-tuning, the lack of good performance for different model dynamics can be explained by the fact that these fuzzy inference schemes need better rules and membership functions to represent the varying dynamics of the motor model.

## 10 Concluding remarks

In this case study, we have shown that self-tuning of PID gains using fuzzy and neural methods can be quite successful in improving a PID controller's performance. The fuzzy self-tuning scheme using TS FIS shows much promise. Using only six rules and relatively simple changes compared to the conventional controller, it manages to improve the performance significantly, in particular for a control signal disturbance. It also adapts well to changes in the motor inertia. Tuning for the Sugeno FIS went much more smoothly because there was more control of the controller's behavior in the different regions of operation.

The neural network-based self-tuning scheme performs best on the nominal model and its performance fares very well in input disturbance rejection. However, for different model dynamics, the neural network-based self-tuning scheme needs more training to get better generalization of plant dynamics. Overall, the performance of fuzzy logic and neural network-based self-tuning of PID controller gains seems to give promising results and further work is needed in knowledge base optimization and learning to get even better results.

## References

1. Fleming, P.J., Ruano, A.E.B., and Jones, D.I. (1992) "Connectionist approach to PID auto-tuning," *IEE Proceedings-D*, vol. 139, pp. 29–285.
2. Akhyar, S., and Omatsu, S. (1993) "Neuromorphic self-tuning PID controller," *Proc. of 1993 IEEE ICNN*, pp. 552–7.
3. Astrom, K.J., and Hagglund, T. (1988) *Automatic Tuning of PID Controllers*, Instrument Society of America, Research Triangle Park, NC.
4. Astrom, K.J., and Hagglund, T. (1993) "Automatic tuning and adaptation for PID-controllers – a survey," *IFAC J. Contr. Eng. Practice*, vol. 1, pp. 699–714.
5. Cameron, F., and Seborg, D.E. (1983) "A self-tuning controller with a PID structure," *International Journal of Control*, vol. 30, pp. 401–17.
6. Deshpande, P.B. (1991) "Improved quality control on line with PID controllers," *Chem. Eng. Progress*, pp. 71–6.

7. Psaltis, D. *et al.* (1988) "A multilayered neural network controller," *IEEE Control System Magazine*, vol. 8, pp. 17–21.
8. Jeong, K.C. *et al.* (1998) "A fuzzy logic-based gain tuner for PID controllers," *IEEE International Conference on Fuzzy Systems*, pp. 551–4.
9. Gawthrop, P.J. (1986) "Self-tuning PID controllers: algorithm and implementation," *IEEE Transactions on Automatic Control*, vol. 31, pp. 201–9.
10. Goldberg, D.E. (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA.
11. Li, Y.F., and Lau, C.C. (1989) "Development of fuzzy algorithms for servo systems," *IEEE Control Systems Magazine*, pp. 65–71.
12. Michalewicz, Z. (1996) *Genetic Algorithms + Data Structures = Evolution Programs*, Springer Verlag, New York.
13. Morari, M., and Zafiriou, E. (1989) *Robust Process Control*, Prentice-Hall, Englewood Cliffs, NJ.
14. Omidvar, O., and Elliott, D.L. (1997) *Neural Systems for Control*, Academic Press, San Diego, CA.
15. Passino, K.M., and Yurkovich, S. (1998) *Fuzzy Control*, Addison-Wesley Longman, California.
16. Tan, S., He, S.Z., and Xu, F.L. (1993) "Fuzzy self-tuning of PID controllers," *Fuzzy Sets and Systems*, vol. 56, pp. 37–46.
17. Saerens, M., and Soquet, A. (1991) "Neural controller based on backpropagation," *IEE Proceedings-F*, vol. 138, pp. 55–62.
18. Swiniarski, R.W. (1990) "Novel neural network-based self-tuning PID controller which uses pattern recognition technique," *Proceedings of the American Control Conference*, pp. 3023–4.
19. Tsoukalas, L.H., and Uhrig, R.E. (1997) *Fuzzy and Neural Approaches in Engineering*, John Wiley & Sons, New York.
20. Ying, H. (1998) "Constructing nonlinear variable gain controllers via the Takagi-Sugeno fuzzy control," *IEEE Transactions on Fuzzy Systems*, pp. 226–34.
21. Tomikusa, M., Zhao, Z.Y., and Isaka, S. (1993) "Fuzzy gain scheduling of PID controllers," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 23, pp. 1392–8.
22. Ziegler, J.G., and Nichols, N.B. (1943) "Optimum setting for automatic controllers," *Trans. ASME*, vol. 65, pp. 433–44.

## Case study 2

### Stabilizing control of a high-order power system by neural adaptive feedback linearization

© 2003 IEEE. Reprinted, with permission, from “*Stabilizing Control of a High-Order Power System by Neural Adaptive Feedback Linearization*,” K. Fregene and C. Kennedy, *IEEE Transactions on Energy Conversion*, vol. 18, no. 1, March 2003, pp. 149–56.

## 1 Introduction

The differential geometric technique of state feedback linearization has been well investigated for the purpose of power system control [1, 2]. The main idea is to perform a coordinate transformation of the nonlinear system equations and define a “new” control input so that in the new coordinates, the nonlinearities in the plant are either wholly or partially masked. This formulation often results in linearizations which are valid for larger practical operating points of the power system than local Jacobian linearizations about particular points. Feedback Linearizing Control (FBLC) has been applied to power systems in two main ways. In the *input-state* FBLC, the system state becomes a linear function of a new control input and a new state (the output typically remains a nonlinear function). *Input-output* FBLC is formulated such that the output is a linear function of a new control input. Owing to the complex mapping carried out during the FBLC process, the common practice in power systems is to use a reduced 3rd order model rather than the more realistic 7th order model (for a single machine system). As shown in [1, 3], issues of robustness and effectiveness of control, among others, are brought to the fore when reduced-order FBLCs are applied to this more realistic system model. Additionally, FBLC requires exact knowledge of the system dynamics, which may not be attainable in a real power system. For general linearizable systems, adaptive FBLC has been proposed to remedy this situation (see [4], for instance), but the formulation often assumes nonlinearities which can only be parameterized linearly.

These problems are significantly mitigated by using multilayer neural networks (NNs).

- (i) The NN provides greater flexibility in parameterizing the nonlinear power system dynamics.
- (ii) The requirement for exact knowledge of the system dynamics, full state measurement as well as other difficulties associated with FBLC for power systems is avoided in this approach.

- (iii) Formulating the FBLC mapping in terms of the NN weights simplifies the problem to that of designing a linear pole-placement controller which is itself not a neural network but is adaptive in the sense that the neural system model can be adapted on line.

In our approach, two NNs are trained off line to model the power system. The FBLC mapping is done using this NN model and adapted on line in response to changes in system parameters and operating conditions. This allows both an excitation controller and a power system stabilizer to be easily designed by using conventional linear control tools. Off line training is accomplished in batch form using the Levenberg–Marquardt optimization as in [5] while on line adaptation proceeds incrementally via the ubiquitous backpropagation algorithm [6]. In this second respect, the adaptive FBLC scheme is related to that presented in [7].

There are several features that are noteworthy about this NN-based method. Unlike many analytic FBLC for power systems, an NN-based approach allows us to use a more realistic high-order model of the power system. Moreover, the fact that the weights of the NN identifiers are adapted on line means that, at each instant, the control law is computed from current estimates of the plant. The approach therefore keeps track of the operating conditions of the system and responds to changes in system parameters appropriately. Finally, and in a broader sense, this formulation shows that it is possible to use proven tools from linear control theory in the design of NN-based stabilizing controllers for power systems.

This case study is organized as follows. Section 2 presents the single generator system used in our study. The system's *relative degree* and *minimum phase* properties, important for feedback linearization to be successfully applied, are also discussed. In addition, we motivate the choice of a neural structure for modelling by identifying the class of systems to which this power system belongs. Section 3 focusses on the neural modelling of the generator while section 4 presents a systematic design of the excitation controller. This design is modified slightly to achieve power system stabilizer functionality in section 5. Finally, simulation results are presented and discussed in section 6.

## 2 The system model

The model used is developed in [8]. It is a 7th order nonlinear single generator connected through a balanced pair of transmission lines to an infinite bus. Figure 10.17 shows a single generator (with the excitation and stabilization accessories) connected to an infinite bus. In the figure,  $Z_{tr}$  represents the transformer,  $V_\infty$  represents the voltage at the infinite bus while the  $Z_{line}$  pairs represent the transmission lines.

The state variables  $x := [x_1, \dots, x_7]^T$  are defined as follows:

$$x := [\delta \omega \lambda_d \lambda_q \lambda_f \lambda_{kd} \lambda_{kq}]^T$$

and the system model is given by

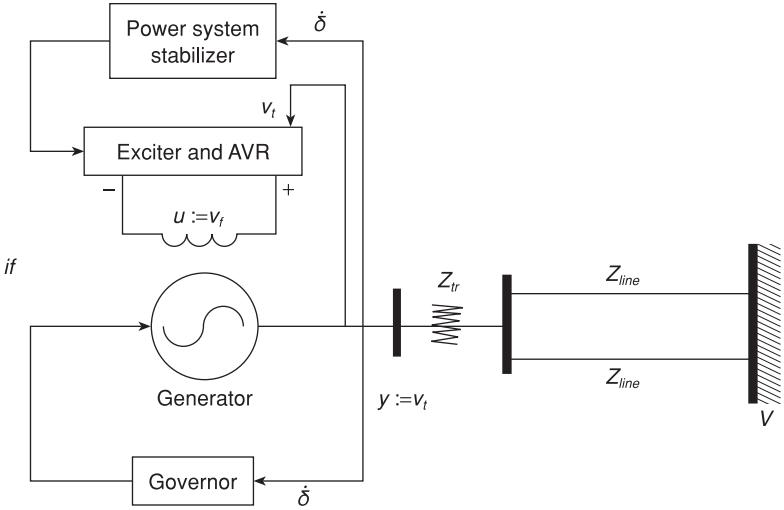


Figure 10.17: Single generator/infinite bus configuration

$$\dot{\delta} = \omega$$

$$\dot{\omega} = \frac{\omega_b}{2H} [P_m - P_e(\delta, \lambda_d, \lambda_q) - D\omega] \quad (1)$$

$$\dot{\lambda} = \omega_b ([RL^{-1} + Z]\lambda + [v_d v_q v_f 0 0]^T),$$

where  $\lambda := [\lambda_d \ \lambda_q \ \lambda_f \ \lambda_{k_d} \ \lambda_{k_q}]^T$ , are the per unit flux linkages,  $R := \text{diag}[r_s r_s - r_f - r_{k_d} - r_{k_q}]$ , are per unit resistances.  $H$  is the inertia constant,  $D$  encapsulates the damping in the system,  $L$  is a  $5 \times 5$  matrix of inductances and  $Z$  is a  $5 \times 5$  constant matrix.

The control input ( $u$ ) is the field voltage  $v_f$  and the output to be controlled ( $y$ ) is the generator terminal voltage  $v_t$  given by

$$y = v_t = h(\delta, \lambda_d, \lambda_q) = \sqrt{v_d^2 + v_q^2}.$$

$P_m$  and  $P_e$  are respectively the mechanical and electrical power with  $P_e = v_d i_d + v_q i_q$ . Here,  $v_{q,d}$  and  $i_{q,d}$  are  $q-$  and  $d-$  axis voltages and currents respectively.

The system (1) belongs to the class of control affine nonlinear systems of the form

$$\begin{aligned} \dot{x} &= f(x) + g(x) \cdot u \\ y &= h(x). \end{aligned} \quad (2)$$

with  $f$  and  $g$  being smooth functions. The state  $x \in \mathbb{R}^n$ , input  $u \in \mathbb{R}$  and output  $y \in \mathbb{R}$ .

Because the system has unit relative degree and is minimum phase, the identification structure used is:

$$\begin{aligned} y[k+1] = & f[y(k), \dots, y(k-n+1), u(k-1), \dots, u(k-m)] \\ & + g[y(k), \dots, y(k-n+1), u(k-1), \dots, u(k-m)] \cdot u(k) \end{aligned} \quad (3)$$

where  $y$  is the output and  $u$  is the input (for the 7th-order generator with relative degree 1,  $n = 7$  and  $m = 6$ ).

### 3 Modeling the power system in NNs

The set-up to model the power system as NNs according to the structure (3) is shown in Figure 10.18. *TDLs* are tapped delay lines used to implement the regressor structure for both networks. *NN-1* and *NN-2* are feedforward neural networks whose input vector consists of present output and delayed inputs and outputs used to predict the next output  $y(k+1)$ . *NN-1* and *NN-2* each has one hidden layer made up of a total of  $p$  and  $q$  neurons respectively with hyperbolic tangent *tanh* activation function and an output layer of one neuron with linear activation function. All weights are initialized randomly and contain biases.

A training set is generated by exciting the plant with a variety of input signals  $u$  (of appropriate range) and measuring the output  $y$ . That is, let  $Z^N = \{[u(k), y(k)] | k = 1, \dots, N\}$ , represent the training data pairs and  $N$  be the total number of data sets. The estimate of the plant output given by the neural nets is

$$\hat{y}[k+1] = \hat{f}[z(k), w] + \hat{g}[z(k), v] \cdot u(k) \quad (4)$$

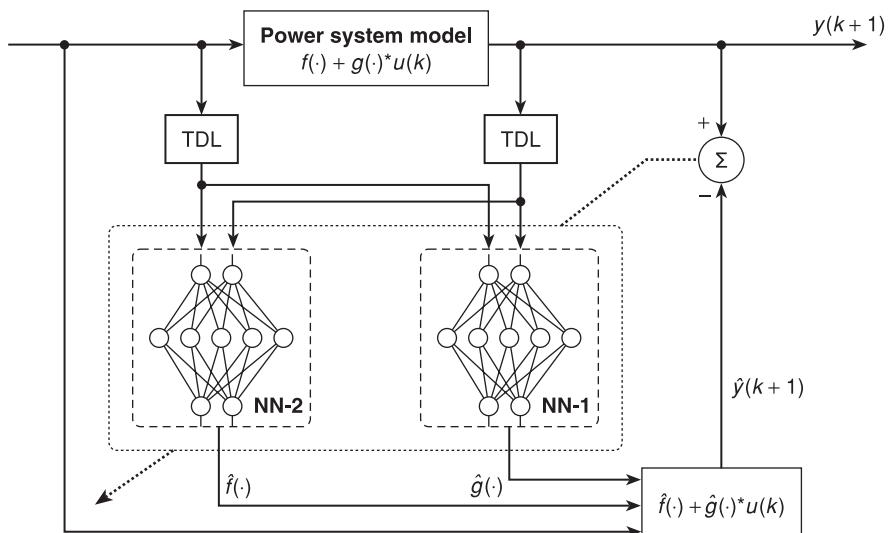


Figure 10.18: Neural system identification of the generator

where

$$\begin{aligned}\hat{f}[z(k), w] &= \sum_{i=1}^p w_i \tanh \left( \sum_{j=1}^{m+n} w_{ij} z_j + b_i \right) + b \\ \hat{g}[z(k), v] &= \sum_{i=1}^q v_i \tanh \left( \sum_{j=1}^{m+n} v_{ij} z_j + a_i \right) + a;\end{aligned}$$

$(w, b)$  and  $(v, a)$  are weights and biases respectively for the  $\hat{f}(\cdot)$  and  $\hat{g}(\cdot)$  networks. Let the set of possible weights be  $\hat{\Theta}$  and define the cost function  $J$  as the mean square error in predicting the plant output. That is,  $J_N(\Theta, Z^N) =$

$$\frac{1}{2N} \sum_{k=1}^N [y(k+1) - \hat{y}(k+1|\Theta)]^T [y(k+1) - \hat{y}(k+1|\Theta)]. \quad (5)$$

The optimal weights are determined by solving

$$\hat{\Theta} = \arg \min_{\Theta} J_N(\Theta, Z^N)$$

using the update rule  $\Theta_{k+1} = \Theta_k + \rho_k S_k$ ; where  $\Theta_k$  is the current iterate of the set of weights,  $\rho_k$  is the step size and  $S_k$  is the search direction.

For identification purposes, the weight update is done in batch form using the Levenberg–Marquardt optimization as described in [5]. When the cost function (5) is minimized to an acceptable level, weight adaptation is discontinued and the neural model is validated on the training data set and subsequently cross-validated on other sets of data not in the original training set.

## 4 Adaptive linearizing control design

The proposed scheme works by using the neural estimates of the generator model as the basis for synthesizing a FBLC. In this case, the excitation controller/stabilizer is of the simple pole-placement genre. Details of the development of such a controller are discussed in [3].

Essentially, if  $r(k)$  is some reference to be tracked, then defining the input to the generator as

$$u(k) = \frac{1}{g[z(k)]} [-f[z(k)] + r(k)] \quad (6)$$

approximately masks the nonlinearities so that the mapping from  $r(k)$  to the system output is linear. Our FBLC design is predicated on appropriately modifying the  $r(k)$  in (6) in order to obtain desired system response. To this end, we define a virtual control input  $\beta(k)$  to the feedback linearized system which is in fact a modification of the reference. Figure 10.19 illustrates the

mapping from this virtual control input to the actual input (in this case, the field voltage) of the generator for our particular application. In general, this relationship is given by

$$u(k) = \frac{1}{g[\mathbf{z}(k)]} [-f[\mathbf{z}(k)] + \beta(k)], \quad (7)$$

which results in input–output closed loop dynamics of

$$y[k+1] = \beta(k).$$

For the primary objective of tracking a reference voltage  $v_{ref}$ , a suitable choice of the control  $\beta(k)$  is

$$\beta(k) = K_1 v_{ref}(k) - [C_{p-1}y(k) + C_{p-2}y(k-1) + \dots + C_0y(k-p+1)] \quad (8)$$

with  $p \in \mathbb{Z}^+$  and  $C_i \in \mathbb{R}$ . Therefore,

$$y[k+1] + C_{p-1}y(k) + C_{p-2}y(k-1) + \dots + C_0y(k-p+1) = K_1 v_{ref}(k).$$

Notice that the transfer function between the reference and  $y$  is given by

$$\frac{V_t[z]}{V_{ref}[z]} = \frac{Y[z]}{V_{ref}[z]} = \frac{K_1}{z^p + C_{p-1}z^{p-1} + \dots + C_1z + C_0} \quad (9)$$

Therefore closed loop poles for the system can be assigned by selecting constants  $C_i$ 's so that the poles of (9) are all in the unit circle.

Using NN estimates of the functions  $f$  and  $g$  given by:

$$\hat{y}(k+1) = \hat{f}[\mathbf{z}(k), \mathbf{w}] + \hat{g}[\mathbf{z}(k), \mathbf{v}] \cdot u(k), \quad (10)$$

rather than the actual functions (which are not guaranteed to be known precisely for such a highly complex nonlinear system), it is apparent from Funahashi's theorem of universal approximation [10], that  $f$  and  $g$  can be approximated as closely as possible by a suitably parameterized NN. Accordingly, the purpose of the “off line” training phase is to obtain good initial estimates  $\hat{f}$  and  $\hat{g}$  for this purpose.

The control input  $u(k)$  is then given as:

### Control Law:

$$u(k) = \frac{-\hat{f}[\mathbf{z}(k), \mathbf{w}(k)] + \beta(k)}{\hat{g}[\mathbf{z}(k), \mathbf{v}(k)]} \quad (11)$$

in which  $\beta(k)$  is as previously defined in (8) and  $[\mathbf{z}(k), \cdot]$  represents the regressor structure parameterized as weights of the NN. The control  $u(k)$  is applied to

both the plant and the NN model; the network weights are updated using the error between the plant output and the output predicted by the neural model. This error ( $e(k)$ ) (together with a *deadzone* of radius  $d_0$ ) is used in the on line weight update rule.

## 5 Power system stabilizer

The *power system stabilizer* adds damping to the generator rotor angle oscillations by modulating its excitation using auxiliary stabilizing signal(s). For the machine model being used, a suitable signal for modulating the excitation is the rotor speed  $\omega(k)$ . The linearizing control law (7) is therefore modified to introduce a torque component which damps out the oscillations of the rotor angle as follows:

$$u(k) = \frac{\beta(k) - \hat{f}(\cdot)}{\hat{g}(\cdot)} + G_{pss} \cdot \omega(k) \quad (12)$$

where  $G_{pss}$  is implemented as a discrete time transfer function of low order to essentially mimic the damping provided by a standard IEEE PSS1A [11]. The gain component of  $G_{pss}$  is selected using a root locus criterion discussed in [1]. The proposed excitation controller/power system stabilizer control set-up is shown in Figure 10.19.

## 6 Simulation results and discussions

### 6.1 Identification

To accomplish the system identification part, the power system is modelled in SIMULINK and sampled at 2 ms to ensure the fast electrical dynamics of the

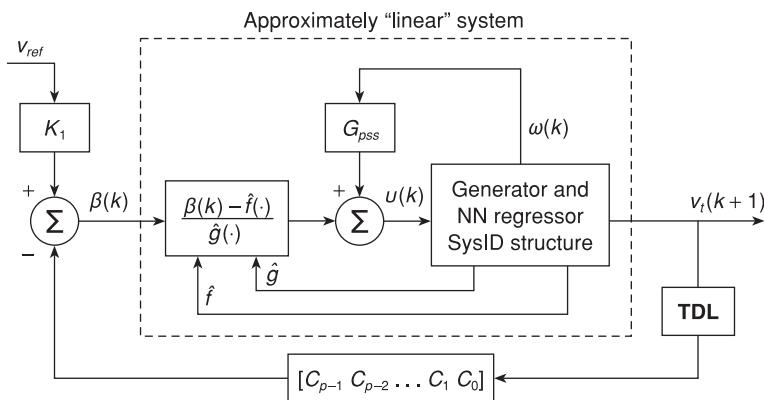
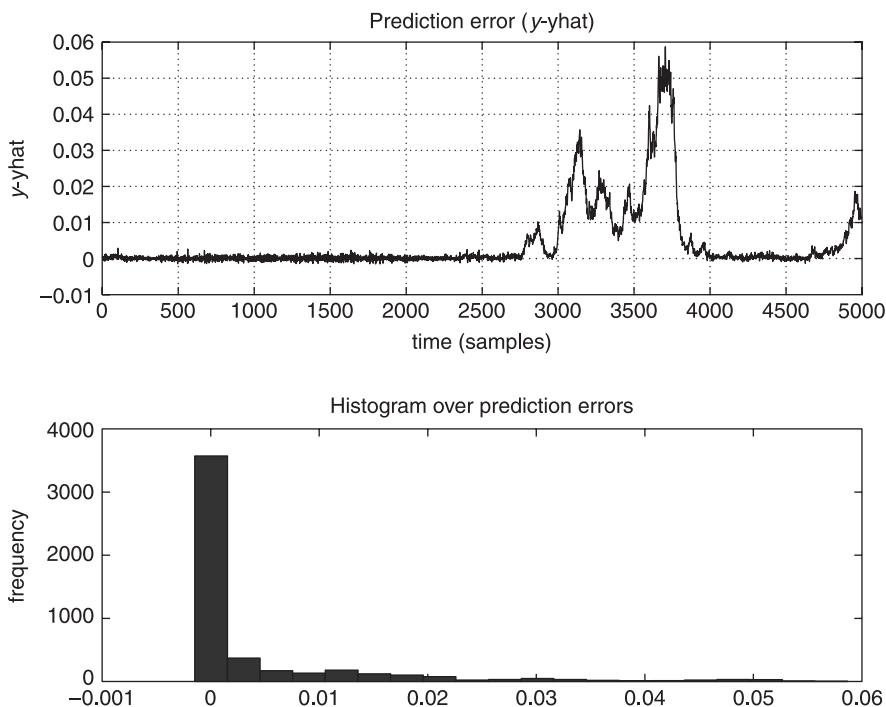


Figure 10.19: Adaptive linearizing controller/stabilizer set-up



**Figure 10.20:** Prediction error  $e$  on test data (top) and corresponding histogram (bottom)

plant are adequately captured. Next, random input signals  $v_f \in [-0.1, 0.1]$  are used to excite the plant and the terminal voltage  $v_t$  is measured for each case. 10,000 such data pairs  $(v_f, v_t)$  are collected and divided into two parts – one for training the neural nets and the other for cross-validating the resulting neural model. Structural details of the NN are:  $\hat{f}(\cdot)$  and  $\hat{g}(\cdot)$  both have 5 hidden hyperbolic tangent neurons with bias and one linear output neuron; the weight matrices for both  $\hat{f}(\cdot)$  and  $\hat{g}(\cdot)$  have dimension  $5 \times 14$  in the hidden layer and  $1 \times 6$  in the output layer; the input to each network consists of current and delayed  $v_f$  and  $v_t$  up to the most delayed signal in each case (13 in all) with the bias term accounting for the last column of the weight matrix.

The optimization routine used for off line weight adaptation is allowed to proceed for a total of 150 iterations at the end of which the cost function is minimized to the order of  $10^{-6}$ . Figure 10.20, which shows the prediction error from cross-validation, was generated using tools from [5]. Observe from Figure 10.20 that the error is essentially close to zero for most of the samples (this is indicated prominently on the histogram) but is more pronounced around the range 3000–4000; even in this range, the errors are not unduly large. The nature and distribution of the errors motivate a choice of the deadzone radius as  $d_0 = 0.01$  for on line adaptation. It will soon become apparent that on line weight adaptation serves to keep the errors well within bounds during the control action.

## 6.2 Excitation control and stabilization

In our simulation studies, we compare the proposed control scheme with an AVR/exciter model ST1A and conventional PSS1A from IEEE Standard 421.5 [11]; and also with an analytic FBLC based on the 3rd order model of the single machine system as presented in [1].

The constants  $C_i$  for  $\beta$  in (8) and (12) are obtained from the characteristic polynomial of the desired closed loop system (9).  $K_1$  is chosen to be  $1 + \sum_{i=0}^{p-1} C_i$ , to yield unity DC gain,  $p = 7$  and poles are placed at  $z = 0.7$ . The transfer function is then given by

$$\frac{V_t[z]}{V_{ref}[z]} = \frac{K_1}{z^7 + C_6 z^6 + \dots + C_0}$$

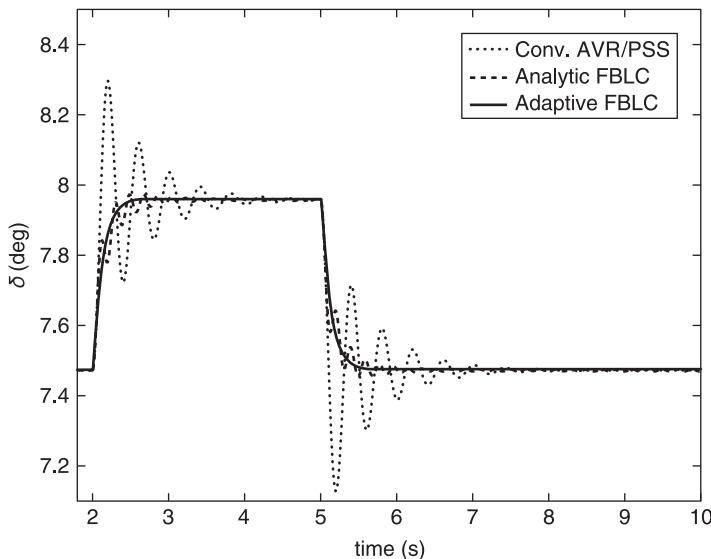
The simulation scenarios considered are as follows:

### 6.2.1 Light load test

Here, a  $0.1 \text{ p.u.}$  step increase in the mechanical power  $P_m$  takes place at  $2\text{s}$  with a return to the initial condition  $3\text{s}$  later. The infinite bus has  $0.8 \text{ pf}$  (*lag*). and  $P_m$  is originally  $1.6212 \text{ p.u.}$  The response is shown in Figure 10.21.

### 6.2.2 Voltage reference change test

A  $0.1 \text{ p.u.}$  step change in exciter reference voltage takes place at  $5\text{s}$  and the system then returns to its initial condition after  $2.5\text{s}$ . The terminal voltage and power angle response are shown in Figure 10.22 and Figure 10.23



**Figure 10.21:** Power angle response to a  $0.1 \text{ p.u.}$  increase in  $P_m$

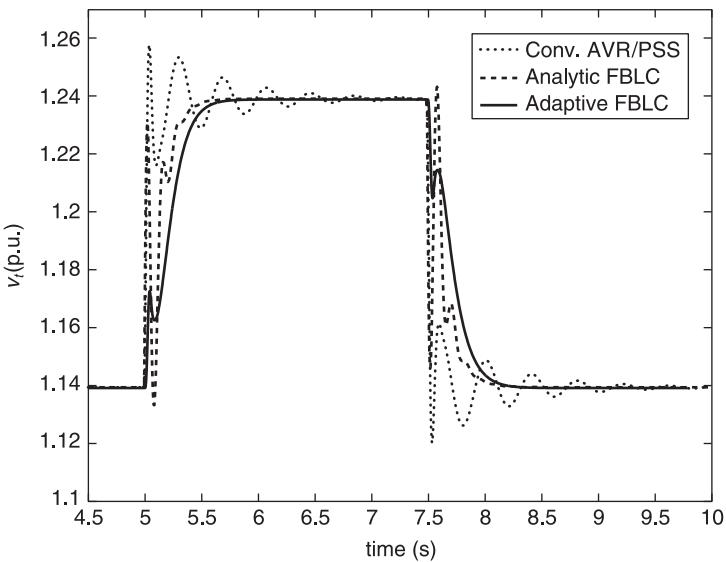


Figure 10.22: Voltage dynamics for a 0.1 p.u. step increase in  $v_{ref}$

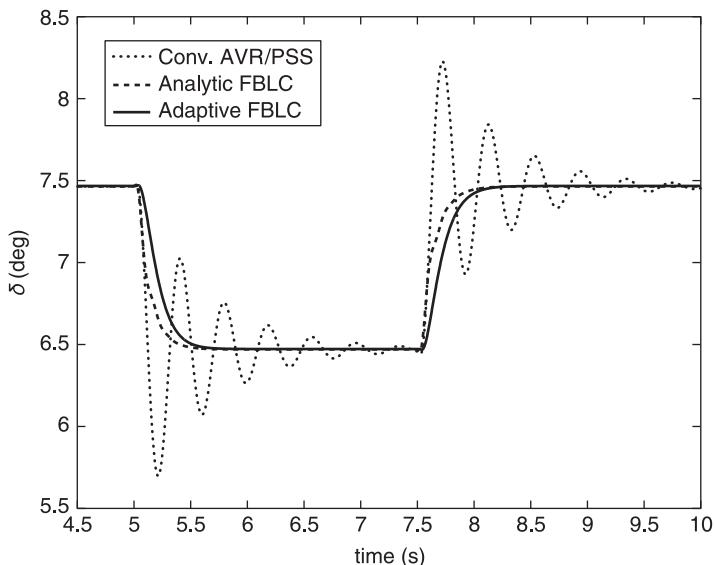
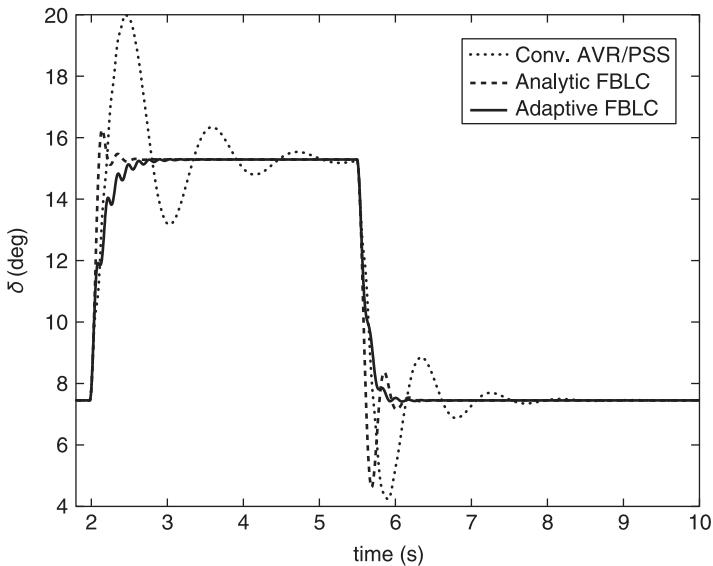


Figure 10.23: Power angle response to a 0.1 p.u. step increase in  $v_{ref}$



**Figure 10.24:** Power angle response during transient test

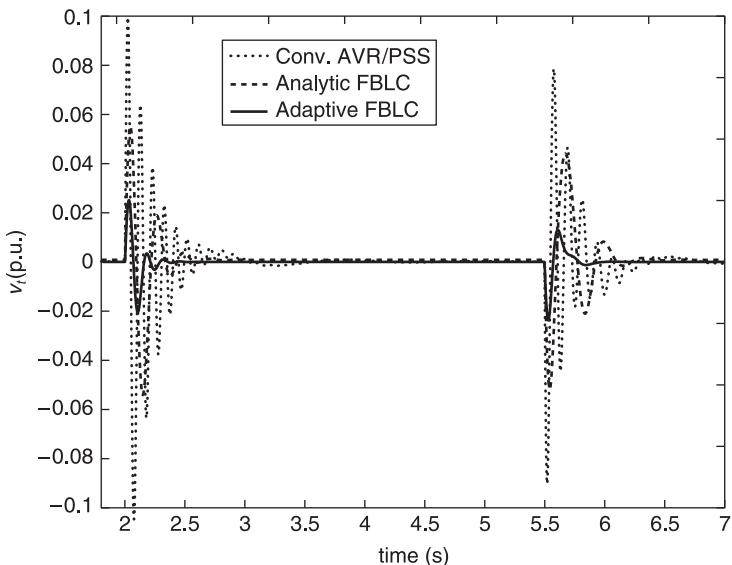
respectively. The adaptive FBLC exhibits no overshoot for the AVR functionality, although it is not markedly different from the analytic FBLC in the damping of power angle oscillations.

### 6.2.3 Transient test

A three-phase to ground short circuit fault is applied at one of the transmission lines at  $t = 2\text{ s}$  and then cleared half a second later. Reclosure is successfully accomplished about 3s later. Figure 10.24 shows the behavior of the power angles while the control effort required for each of the controllers tested is depicted in Figure 10.25. Observe that the level of control required by the adaptive FBLC scheme is several orders of magnitude less than both the analytic and the conventional controllers. Also, less harmonics are visible in the adaptive FBLC. This suggests that the weights are adapted rather rapidly to the operating conditions. To confirm this reasoning, changes in both the  $\hat{f}$  and  $\hat{g}$  networks' weights are monitored as discussed in subsection 6.2.5.

### 6.2.4 Stability margin

We increase  $P_m$  slowly from its initial value (1.6212 *p.u.* and 0.8 *pf (lag)*) till the conventional AVR/PSS, analytic FBLC and adaptive FBLC all begin to go unstable. The percentage of increase in  $P_m$  before each goes unstable and the corresponding maximum rotor angles are given in Table 10.3. The initial rotor angle is  $7.5^\circ$ .



**Figure 10.25:** Control effort required during transient test

**Table 10.3:** Table of stability margins for all controllers

	Conv.	Analyt. FBLC	Adapt. FBLC
$P_{m_{max}}$ %	33.04	52.61	59.68
$\delta_{max}$ deg.	76.3	91.63	102.9

### 6.2.5 On line weight adaptation

In order to study the effect of transients on on line weight adaptation, the weights values for both the  $\hat{f}$  and  $\hat{g}$  networks were monitored during the transmission line short circuit fault. The weights changed significantly to provide instantaneous estimates of the changing system parameters used in the control synthesis. After recovery from the fault condition, the weights gradually returned to steady values, which were not exactly identical to the pre-fault network weights.

## 7 Concluding remarks

An adaptive feedback linearizing excitation controller/stabilizer which is based on NNs has been described for a high-order single machine power system. The controller provides tracking performance with less high frequency harmonics than both a conventional and an analytic nonlinear FBLC based on a reduced-order model of the system. For voltage tracking and power angle stabilization, the proposed controller requires less control effort than

the analytic and conventional controllers. It also boosts the system's stability margin and provides superior transient response. Advantages of this approach lie in the fact that exact knowledge of the power system dynamics is not absolutely necessary, the system state need not all be measurable and familiar tools from linear systems theory are applicable in the design methodology. Also, the complicated mathematical mapping required to carry out analytic feedback linearization is avoided.

## References

1. Kennedy, D., Miller, D., and Quintana, V. (1998) "A nonlinear geometric approach to power system excitation control and stabilization," *Electric Power and Energy System*, vol. 20, no. 8, pp. 501–15.
2. Chapman, J.W., Ilić, M.D., King, C.A., Eng, L., and Kaufman, H. (1993) "Stabilizing a multi-machine power system via decentralized feedback linearizing excitation control," *IEEE Trans. Power System*, vol. 8, pp. 830–9.
3. Fregene, K. (1999) "Neural adaptive feedback linearizing control of a high-order power system," M.S. thesis, Dept. Elect. and Computer Eng, Univ. Waterloo, Ont., Canada.
4. Kanellakopoulos, I., Kokotović, P.V., and Morse, A.S. (1991) "Systematic design of adaptive controllers for feedback linearizable systems," *IEEE Trans. Automatic Contr.*, vol. 36, no. 11, pp. 1241–53.
5. Nøgaard, M. (1997) "Neural network based system identification toolbox," Tech. Rep. 97-E-851, Dept. of Automation, Technical University of Denmark.
6. Rumelhart, D., Hinton, G.E., and William, R.J. (1986) "Learning internal representations by error propagation," in *Parallel Distributed Processing*, D.E. Rumelhart and J.C. McClellan (eds), vol. 1, MIT Press.
7. Chen, F-C., and Khalil, H.K. (1995) "Adaptive control of a class of nonlinear discrete-time systems using neural network," *IEEE Trans. Automatic Contr.*, vol. 40, no. 5, pp. 791–801.
8. Kundur, P. (1994) *Power System Stability and Control*, McGraw-Hill, pp. 54–135.
9. Isidori, A. (1989) *Nonlinear Control Systems*, Springer-Verlag, 2nd edn.
10. Funahashi, K. (1989) "On the approximate realization of continuous mappings by neural networks," *Neural Networks*, vol. 2, pp. 183–92.
11. IEEE Standard 421.5 (1992) "IEEE Recommended Practice for Excitation Systems for Power System Stability Studies."

## Case study 3

### Soft computing tools for solving a class of facilities layout planning problem

© 2003 IEEE. Extracts reprinted, with permission, from “*Computational intelligence tools for solving the facilities layout planning problem*,” Karray, F., Zanddin, E., Hegazy, T., Shabeb, A., Elbeltagi, E., *Proceedings of the 2000 American Control Conference*, vol. 6, 28–30 June 2000, pp. 3954–58.

## 1 Introduction

The layout of temporary facilities in a construction site deals with the selection of their most efficient positioning in order to operate efficiently and cost effectively. The layout design seeks the best arrangement of facilities within the available area. In the design process of the layout, many objectives must be considered to effectively utilize people, equipment, space, and energy. This study proposes a soft computing-based approach to improve the layout process of facilities. The main objective is on obtaining the closeness relationship values between each pair of facilities in a construction site. To achieve this, an integrated approach, using the fuzzy set theory and genetic algorithms, is used to investigate the layout of temporary facilities in relation to the planned building(s) in a construction site. An example application is presented to illustrate the proposed approach and the results are then discussed. Depending on the importance of relationships among the various facilities in the construction site, this study is expected to provide engineers with an appropriate tool to compare and evaluate different layouts and select the most appropriate one.

## 2 Background

The layout problem is one of the most crucial in the planning of service facilities, since efficient layout is critical to cost-effective operation of these facilities. Despite its importance, site planning is often neglected and the attitude of engineers has been that it will be executed as the project progresses. Good facility layout, however, is important to promote safe and efficient operations, minimize travel time, decrease material handling, and avoid obstructing material and equipment movements (Tommlein *et al.* 1992). In the design process of the layout, therefore, many objectives must be considered to effectively utilize people, equipment, space, and energy (Tompkins and White 1984). A layout is traditionally developed using relationships among the various facilities. These relationships are based on the judgement

of experts who decide the importance of such relationships between each pair of facilities. The decision of experts, however, is vague and is usually based on many quantitative or qualitative considerations pertaining to the closeness relationships among the facilities.

A large amount of work has been reported in the literature to deal with the site layout problem. Due to the complexity of the problem, mathematical optimization was only successful for a very limited number of facilities (Rodriguez-Ramos and Francis 1983; Tommelein *et al.* 1992). This has contributed to the development of various tools that use non-traditional techniques based on Artificial Intelligence and heuristic approaches. The use of artificial neural networks, for example, was proposed by Yeh (1995) to optimize a pre-determined site layout. Most researchers, however, have used heuristic approaches and knowledge-based systems to layout the facilities on a construction site (e.g., CONSITE: Hamiani 1987, SightPlan: Tommelein 1989). Other researchers have also dealt with the layout problem from different perspectives (Seehof and Evans 1967, Lee and Moore 1967, Francis and White 1974).

Most of these efforts, however, use some simplified assumptions and do not consider the “vagueness” and natural language that humans use to control complex systems such as facilities planning. On the other hand, they use only a rectangular site shape and rectangular facilities with a specific aspect ratio. Also, heuristic and knowledge-based approaches are not guaranteed to provide optimum solutions and usually do not include an objective function to evaluate alternative layout solutions. In an attempt to overcome these limitations, this paper presents a powerful methodology, based on the fuzzy set theory. To further improve the facilities layout process, genetic algorithms are being used as layout optimization mechanisms. Details of the proposed methodology are described next and an example application is used to illustrate the approach.

### 3 Proposed approach

The layout problem is complex and vague. Complexity generally arises from uncertainty in the form of ambiguity. Problems featuring complexity and ambiguity have been addressed subconsciously by humans since they could think. The fuzzy set theory was introduced by Zadeh (1965) to deal with vague, imprecise, and uncertain problems. Fuzzy set theory has been used as a modelling tool for complex systems, such as facility layout, that are hard to define precisely, but can be controlled and operated by humans. Humans can make decisions in the absence of clearly defined boundaries based on expertise and general knowledge of the task of the system in consideration. The humans' actions are based on the if-then rules they develop over years of knowledge and experience. In addition to the ambiguity associated with facilities layout, the layout problem seeks the best arrangement based on the desired closeness relationships among the facilities. These relationships are generated based on a number of factors. The larger the number of facilities in

a site and the number of factors affecting the design, the more complex the layout problem becomes. The proposed approach is applied to an example application. In this example, it was considered that the construction site consists of five temporary facilities in addition to the building to be constructed. These facilities are:

- (1) Offices, with an area of 160 square meters.
- (2) Warehouse (storage), with an area of 240 square meters.
- (3) Batch plant, with an area of 500 square meters.
- (4) Maintenance workshop, with an area of 120 square meters.
- (5) Quality control laboratories, with an area of 60 square meters.
- (6) The building to be constructed, which consists of two attached buildings with a total area of 2880 square meters.

The construction site, as such, has  $n^*(n - 1)/2 = 6^*(6 - 1)/2 = 15$  possible relationships among all facilities, where  $n$  is the number of facilities in the site. It is assumed that there are three input variables affecting the decision of the planner. These variables are:

- (1) Material flow (MF): defined by the number of parts flowing between each two facilities in the construction site.
- (2) Information flow (IF): defined by the number of communications (oral or reports) between facilities.
- (3) Equipment flow (EF): defined by the number of material handling equipment (trucks, mixers, etc.) used to transfer materials between facilities.

Since the assignment of membership functions is subjective in nature, they can be generated based on humans' experience. In real-world construction projects, project managers, resident engineers, and site superintendents can provide planners with the required data to establish the membership functions of the input variables. In this study, however, membership functions are assumed and are based on the personal judgement of the authors' previous experience in several large-scale construction projects in Asia. Each of the three variables used in this study (MF, IF, and EF) are assumed to have five membership functions as shown in Figure 10.26, Figure 10.27, and Figure 10.28, respectively. If the material flow (MF), for example, is 600 units/day, the flow of material can be expressed as medium, based on the experts' experience (Figure 10.26). Also, each of these variables is assumed to have a weight factor (WF) that affects the closeness between each pair of facilities (Saaty 1980).

Based on the input variables mentioned earlier, the planner assigns a closeness relationship ( $R$ ) to each decision rule as an output variable. This is performed through pair-wise assessments of the relationships between each two facilities. In case where the two facilities are required to be close to each other, in the user's judgement, a high weight value is specified for the two facilities, and vice versa. In the literature (Lee and Moore 1967; Francis and White 1974; Malakooti 1987; Hegazy and Elbeltagi 1999) six closeness

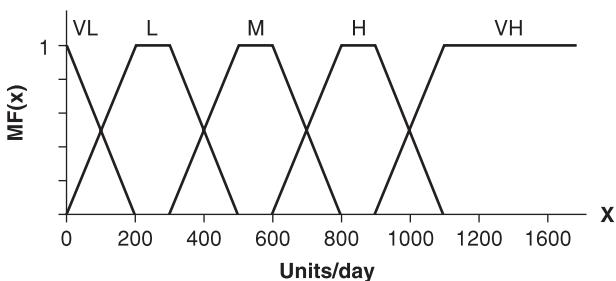


Figure 10.26: Membership functions of material flow (MF)

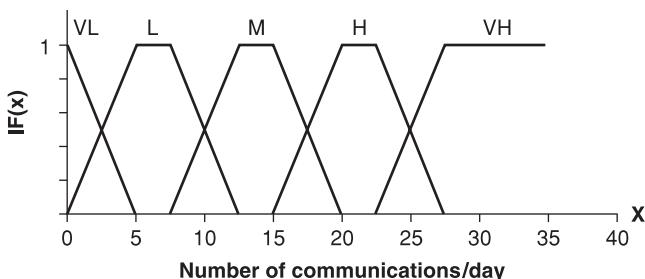


Figure 10.27: Membership functions of information flow (IF)

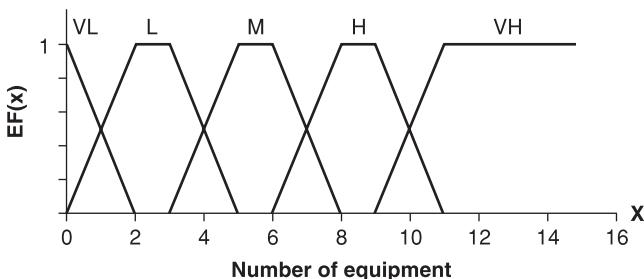


Figure 10.28: Membership functions of equipment flow (EF)

relationships are usually set in advance and the user can give desired weight values associated with each category. The membership functions of the closeness relationships ( $R$ ) are shown in Figure 10.29. A large weight between two facilities means that they share a high level of interaction and, accordingly, the distance between them should be small. A weight value of unity, on the other hand, means that the two facilities have no interaction between them and the distance separating them is irrelevant. It should be noted that the values in Figure 10.29 are presented for illustration only and the planner can set other values based on his own judgement or using a quantitative measure.

The proposed approach is applied on this example using four steps:

**Step 1** Generating the weight factors (WF) for the three linguistic variables between each pair of facilities. Each linguistic variable with its weight factor is used as input to the fuzzy decision-making system.

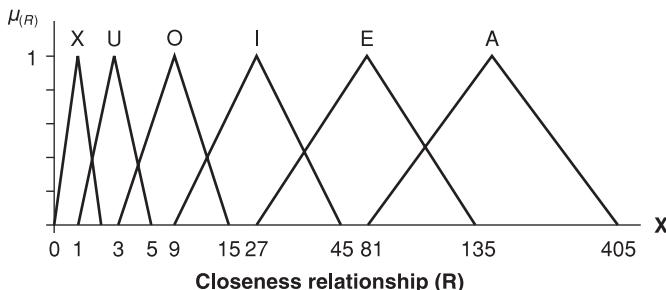


Figure 10.29: Membership functions of closeness relationship (R)

*Step 2* Establishing the *if-then* decision rules.

*Step 3* Generating the facility relationship chart.

*Step 4* Developing the physical layout of the facilities in the construction site.

### 3.1 Generating the weight factors

To illustrate the proposed approach in this example application, the input values for the three linguistic variables are assumed along with the intensity importance of factors for each pair of facilities as shown in Table 10.4. These values are based on the personal judgement of the authors and are needed to generate the overall membership functions.

The numbers shown for the intensity importance of factors ( $a_{ij}$ ) range from 1 for equal importance of factor  $i$  over factor  $j$  to 9 for absolute importance of factor  $i$  over factor  $j$ , where  $a_{ji} = 1/a_{ij}$  (Table 10.5). These numbers represent the true preference of each factor with respect to the other. If the relationship between facilities (1) (offices) and (2) (warehouse) is under consideration, for example, the planner is assigning a value of 2 for the importance of factor 2 over factor 3. This means that there is a range between an equal and a weak importance of factor 2 over factor 3. Considering the relationship between facilities (1) and (6), as another example, the value of 9 assigned for the importance of factor 2 over factor 3 indicates an absolute importance of factor 2 over factor 3.

Considering the relationship between facility (1) (offices) and facility (3) (batch plant), for example, the intensity importance of factors listed in Table 10.4 is used to calculate the weights of factors ( $P_i$ ) shown in Table 10.6. These weights of factors are calculated in three simple steps, as described in Saaty (1980):

*Step 1* Multiplying the  $n$  elements in each row of columns 2, 3, and 4 of Table 10.6 to get the result  $X_i$ , where  $n$  is the number of factors and equals to 3 in this study.

*Step 2* Taking the  $n$ th root of ( $X_i$ ) for each row to get ( $Y_i$ ).

*Step 3* Normalizing by dividing each number ( $Y_i$ ) by the sum of all the numbers ( $\Sigma Y_i$ ).

**Table 10.4:** Relationships and intensity importance of factors among all facilities

Facilities	Fact or 1 (MF)	Fact or 2 (IF)	Fact or 3 (EF)	Intensity importance of factors		
				1 over 2	1 over 3	2 over 3
1-2	0	12	0	0.5	1	2
1-3	0	15	0	0.5	1	3
1-4	0	4	1	1	1	1
1-5	0	23	0	0.25	1	4
1-6	0	30	0	0.112	1	9
2-3	1200	13	10	1.5	1	1
2-4	150	3	2	1	1	1
2-5	100	5	1	2	3	2
2-6	1500	7	4	4	1	0.25
3-4	0	3	3	0.8	0.25	0.33
3-5	400	26	2	0.8	3	3
3-6	1000	5	5	4	1	0.25
4-5	0	2	1	1	1	1
4-6	0	0	2	1	1	1
5-6	5	0	1	1.25	1	0.8

**Table 10.5:** The intensity importance ( $a_{ij}$ )

Intensity importance $a_{ij}$	Definition of importance
1	Equal importance of $i$ and $j$
2	Between equal and weak importance of $i$ over $j$
3	Weak importance of $i$ over $j$
4	Between weak and strong importance of $i$ over $j$
5	Strong importance of $i$ over $j$
6	Between strong and demonstrated importance of $i$ over $j$
7	Demonstrated importance of $i$ over $j$
8	Between demonstrated and absolute importance of $i$ over $j$
9	Absolute importance of $i$ over $j$

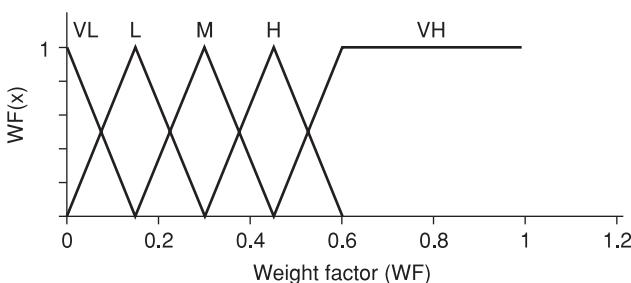
**Table 10.6:** The weight factors between facilities 1 and 3

$i/j$	1	2	3	$X_i = (1 \times 2 \times 3)$	$Y_i (= X^{0.333})$	$P_i = (WF) = Y_i / \sum Y_i$
1	1	0.5	1	0.5	0.794	0.240
2	2	1	3	6	1.817	0.550
3	1	0.3	1	0.33	0.693	0.210
			3			

The result is the weight factor  $P_i$ . This process is repeated for all pairs of facilities to calculate the weight factor ( $P_i$ ) for each variable (Table 10.7). The weight factor is considered one of the inputs that will influence the layout. The higher the weight factor is, the more important the factor becomes when planning the layout. Based on the experience of the authors, the membership functions of the weight factor are assumed as shown in Figure 10.30.

**Table 10.7:** The weight factors ( $P_i$ ) for all variables

Facility	MF	Weight factor ( $P_i$ )		
		IF	EF	
1–2	0.25	0.50	0.25	
1–3	0.24	0.55	0.21	
1–4	0.33	0.33	0.33	
1–5	0.167	0.667	0.167	
1–6	0.091	0.818	0.091	
2–3	0.379	0.289	0.331	
2–4	0.33	0.33	0.33	
2–5	0.54	0.297	0.163	
2–6	0.444	0.111	0.444	
3–4	0.162	0.206	0.632	
3–5	0.397	0.461	0.142	
3–6	0.444	0.111	0.444	
4–5	0.33	0.33	0.33	
4–6	0.33	0.33	0.33	
5–6	0.357	0.286	0.357	

**Figure 10.30:** Membership functions of the weight factor (WF)

### 3.2 Establishing the if-then decision rules

At this point, the fuzzification process is complete. The next step is to establish the decision-making logic (decision rules). These rules usually take the form of *if-then* rules. The number of decision rules between each factor and its weight factor is calculated using the equation:  $NR = (m)^v = (5)^2 = 25$ , where  $NR$  = number of rules,  $m$  = number of membership functions, and  $v$  = number of input variables. Since we are considering three input variables that will affect the decision of the planner, the total number of rules between each two facilities is equal to  $3 \times 25 = 75$  rules (Figure 10.31). For example, if the MF is VL and its WF is VL then the closeness relationship is X (undesirable).

### 3.3 Generating the facility relationship chart (FRC)

A program using a Visual Basic spreadsheet is developed to deal with the inference of the generated *if-then* decision rules. The program can use both

(a) MF and its WF						(b) IF and its WF					
MF/ WF	V L	L	M	H	V H	IF/ WF	V L	L	M	H	V H
VL	X	X	U	O	I	VL	X	X	U	O	O
L	X	U	O	I	I	L	X	U	O	O	I
M	U	O	I	I	E	M	U	O	O	I	I
H	O	I	I	E	E	H	O	O	I	I	E
VH	I	I	E	E	A	VH	O	I	I	E	A

(c) EF and its WF					
EF/ WF	V L	L	M	H	V H
VL	X	X	U	O	I
L	X	U	O	I	I
M	U	O	I	I	E
H	O	I	I	E	E
VH	I	I	E	E	A

X = undesirable  
 U = unimportant  
 O = ordinary  
 I = important  
 E = especially important  
 A = absolutely important

Figure 10.31: If-then rules

the max-min and max-product composition rules of inference. It also allows the user to choose any of the defuzzification methods such as the centroid of the area (COA), mean of maximum (MOM), smallest of maximum (SOM), or the largest of maximum (LOM). In this case study, the max-min composition rule of inference and the centroid of the area (COA) are used for the defuzzification process required to generate the final output crisp values of the facility relationship chart. These values are then used to develop the best layout scheme. This fuzzy decision-making process is repeated for each pair of facilities (i.e., 15 times). The input values for each variable shown in Table 10.4 were entered to the program for each pair of facilities along with its corresponding weight factor to get the overall membership functions and their output crisp values.

The average of output crisp values (closeness relationship) for each pair of facilities is calculated in Table 10.8. These relationships are tabulated in the facility relationship chart (Figure 10.32), which will be used in the next step to generate the facilities' layout in the site.

### 3.4 Developing the physical layout of facilities

Once the facility relationship chart is generated as outlined in the previous subsection, a genetic algorithm-based program (Hegazy and Elbeltagi 1999) is used to generate the layout. The program requires the user to enter the number of facilities to be located in the site, the area of each facility, and the construction site area.

In addition to the temporary facilities, the user is required to enter to the program the fixed facilities in the site (such as roads, building to be

**Table 10.8:** The average closeness relationship values

Facility	MF/WF	IF/WF	EF/WF	Average
1–2	2.60	26.22	2.60	10.48
1–3	2.52	27.01	2.27	10.60
1–4	6.45	8.56	18.70	11.23
1–5	1.62	141.83	1.62	48.35
1–6	1.01	243.01	1.01	81.67
2–3	81.00	9.00	68.81	52.94
2–4	18.66	7.94	19.35	15.31
2–5	23.31	8.95	4.56	12.27
2–6	81.00	2.69	26.63	36.78
3–4	1.59	6.71	81.00	29.77
3–5	23.83	114.10	2.94	46.96
3–6	79.88	2.69	27.00	36.53
4–5	6.53	7.32	18.67	10.84
4–6	6.53	6.45	19.36	10.78
5–6	10.24	2.89	20.76	11.30

Facility name	Area (m <sup>2</sup> )	Facility number	6	5	4	3	2	1
Offices	160	1	81.7	48.4	11.2	10.6	10.5	–
Warehouse	240	2	36.8	12.3	15.3	52.9	–	
Batch plant	500	3	36.5	46.9	29.8	–		
Maintenance workshop	120	4	10.8	10.8	–			
Quality control labs.	60	5	11.3	–				
Building	5120	6	–					

Construction site area = 98.387 m × 84.971 m = 8360 m<sup>2</sup>.

**Figure 10.32:** Facility relationship chart

constructed, etc.). Due to the L shape of the building to be constructed in the site, it is divided into two parts to enable the program to place the building in its required location.

The genetic algorithm-based technique involves three main steps: (1) deciding the objective function; (2) generating an initial population of genes; and (3) selecting an offspring generation. The objective function is generated by multiplying the desired closeness relationship values between each two facilities by the actual distance between them, and summing them up for all facilities. This objective function, therefore, represents the total travel distance for a given site layout. In order to arrive at the optimum layout that brings the least travel distance and, accordingly, to ensure that facilities with higher closeness relationship values will be placed close to each other, the objective function should be minimized. Evaluating the total travel distance

of a given site layout involves determining the centroids of all facilities and then calculating the distances between them ( $d_{ij}$ ). Using the distance calculations, the total travel distance (objective function) of a site layout with  $n$  facilities can, therefore, be calculated as follows:

$$\text{Total travel distance (objective function)} = \sum_{i=1}^{n-1} \sum_{j=i+1}^n d_{ij} R_{ij}$$

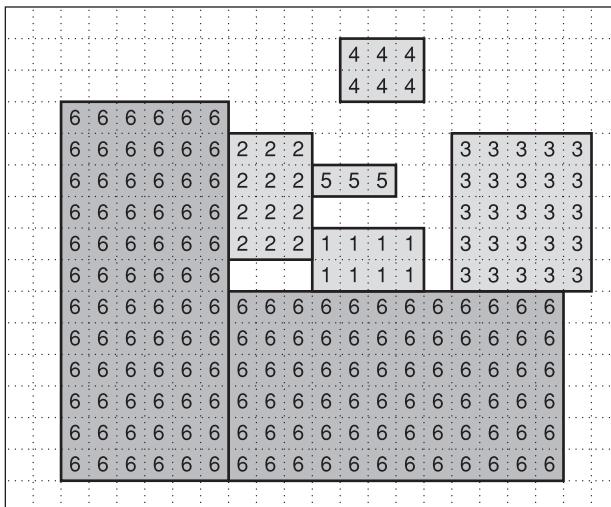
where  $d_{ij}$  is the distance between facilities  $i$  and  $j$  and  $R_{ij}$  is the desired closeness relationship value between facilities  $i$  and  $j$ . A population of parent genes is then generated randomly. In the program, the user is given the flexibility to input the population size (number of genes). Once the population is generated, the fitness of each gene ( $k$ ) in this population ( $m$  genes) is evaluated using the objective function and accordingly its relative merit is calculated as follows:

$$\text{Total fitness} = \sum_{k=1}^m \text{fitness}_k$$

$$\text{Relative merit}_k = \text{fitness}_k / \text{total fitness}$$

As such, the total of the relative merits adds up to 1.0 and the share of each gene is proportional to its relative merit. Finally, the reproduction process among the population members takes place by crossover to produce an offspring. Once an offspring is generated, it is evaluated in turn and can be retained only if its fitness is higher than that of others in the population. This process is continued for a large number of offspring generations until an optimum gene is arrived at. In the program, the user is given the flexibility to input the number of offspring generations. Throughout the process, the entire population soon improves since more fit offspring replace unfit parents. In the present application example, the population size is taken as 200 and the number of offspring generations is taken as 100.

As a first step, the user is required to enter the values related to the closeness relationship between each pair of facilities given in Figure 10.32. The next step is to enter the required population size and the required number of generations per cycle, which are selected as 200 and 100, respectively. Following this, the program generates different solutions (layouts) with their corresponding fitness values (scores). The best solution, basically, is the one with the minimum score (minimum fitness value). In this application example, the best layout is shown in Figure 10.33, which has the minimum score (4061). The shaded area in the figure represents the L-shaped building to be constructed (facility number 6). As shown in the figure, it is noticed that facility number (1) (offices) is placed near the building to be constructed in the site. This is a logical arrangement as compared to real-world situations where offices of engineers and contractors are normally placed near the facility to be constructed. On the other hand, the workshop (facility number 4) is placed far from the building, which is also considered to be a logical placement.



**Figure 10.33:** The selected layout

This is because there is no strong relation between the workshop and the building to be constructed.

#### 4 Comments and discussion

The main feature of the proposed approach that makes it an efficient tool for site layout planning is that it provides different alternative solutions to the site layout and considers the total travel distance as an objective function. It has also been implemented on a commercial spreadsheet program that may be convenient to many practitioners.

On the other hand, one of the drawbacks of the proposed approach is the difficulties encountered in generating the membership functions of the variables affecting the decision of the planner. However, newly developed tools of neuro-fuzzy inferencing (Lin and Lee 1996) have been introduced in recent years and have the potential of automating the process of generating the membership functions from a set of experiential data. Another issue is the problem of establishing the decision rules, which requires a great deal of care and experience. In some particular cases, the planner may not efficiently decide the values for the intensity importance of factors. This is particularly true when the number of facilities increases resulting in complex relationships between the different facilities. Often, however, site planners have adequate experience to overcome such problems.

#### 5 Conclusion

In this study, a construction planning methodology based on the fuzzy set theory is proposed to generate closeness relationship values among pairs of

facilities in a given construction site. This is achieved through the use of a Visual Basic spreadsheet program to utilize its flexible interface and its powerful functions. The program performs the fuzzification of the input variables, generates the fuzzy inference system, and tackles the defuzzification process. The final crisp values (closeness relationships) obtained by the defuzzification process are then used by a genetic algorithms based approach to optimally place facilities within a construction site. An example application is also presented to demonstrate the practicality and potential capabilities of the approach and the results are then discussed. Integrating the proposed approach with newly developed tools of neuro-fuzzy inferencing systems should make it an even more versatile and powerful tool for dealing with facilities layout planning.

## References

1. Francis, R.L., and White, J.A. (1974) *Facility Layout and Location*, Prentice-Hall, Englewood Cliffs, N.Y.
2. Hamiani, A. (1987) "CONSITE: A knowledge-based expert system framework for construction site layout," Ph.D. Thesis, The University of Texas at Austin, Texas.
3. Hegazy, T. and Elbeltagi, E. (1999) "EvoSite: evolution-based model for site layout planning," *Journal of Computing in Civil Engineering*, vol. 13, no. 3, pp. 198–206.
4. Lee, R.C., and Moore, J.M. (1967) "CORELAP-computerized relationship layout planning," *Journal of Industrial Engineering*, vol. 18, pp. 1994–2000.
5. Lin, C-T., and Lee, C.S. (1996) *Neural Fuzzy Systems: A Neuro-fuzzy Synergism to Intelligent Systems*, Prentice-Hall, Englewood Cliffs.
6. Malakooti, B. (1987) "Computer-aided facility layout selection (CAFLAS) with applications to multiple criteria manufacturing planning problem," *Large Scale Systems*, vol. 12, no. 2, pp. 109–123.
7. Rodriguez-Ramos, W.E., and Francis, R.L. (1983) "Single crane location optimization," *Journal of Construction Division, ASCE*, vol. 109, no. 4, pp. 387–397.
8. Saaty, T.L. (1980) *The Analytical Hierarchy Process*, McGraw-Hill, Inc., New York.
9. Seehof, J.M., and Evans, W.O. (1967) "Automated layout design program," *Journal of Industrial Engineering*, vol. 18, pp. 690–695.
10. Tommelein, I.D. (1989) "SightPlan: an expert system that models and augments human decision-making for designing construction site layout," Ph.D. Thesis, Stanford University, Stanford, California.
11. Tommelein, I.D., Levitt, R.E., and Hayes-Roth, B. (1992) "Sight layout modeling: how can artificial intelligence help?" *J. Constr. Engrg. and Mngmt., ASCE*, vol. 118, no. 3, pp. 594–611.
12. Tompkins, J.A., and White, J.A. (1984) *Facilities Planning*, Wiley, New York.
13. Yeh, I.C. (1995) "Construction site layout using annealed neural network," *Journal of Computing in Civil Engineering, ASCE*, vol. 9, no. 3, pp. 201–208.
14. Zadeh, L.A. (1965) "Fuzzy sets," *Information and Control*, vol. 8, pp. 338–353.

## Case study 4

### Mobile position estimation using an RBF network in CDMA cellular systems

## 1 Introduction

In this case study, a radial basis function (RBF) network is used to estimate the location of a mobile station in the coverage area of a code division multiple access (CDMA) cellular network. The technique is based on training an RBF network to estimate the location of a mobile from the measured strength of the signals received by the mobile station from neighboring base stations. We evaluate the performance and show the merit of this method by means of simulation. The possibility of implementing the RBF network in hardware and its speed make this approach appealing for real-time vehicle tracking systems.

## 2 Background

A key difference between wireline and wireless communication systems is the issue of user mobility. In wireline networks, the user location is fixed and can be determined. However, in wireless networks, the user is mobile and does not have a fixed location. The aspect of user mobility has created many challenges in the design and operation of wireless communication systems. One recent challenge emerged from the rapidly growing volume of emergency calls made by mobile subscribers and the failure of automatic determination of the caller location.

In response, the United States Federal Communications Commission (FCC) has required that mobile network operators be capable of accurately determining and providing the location of mobile subscribers that request emergency assistance [1]. The FCC requirements require that the position of emergency callers is determined with a location error that does not exceed 125 m in 67% of the time. In a parallel effort, the European Commission established the Coordination Group on Access to Location Information by Emergency Services (CGALIES) with the mission to set requirements for a pan-European common policy and implementation mechanism that can be applied and used by the European 112 community and emergency service operators [2].

In addition to their need to comply with the regulations set by communication regulatory commissions, mobile network operators can benefit from location information in various applications such as location-based information services, zone-based billing, and network optimization [3], [4].

Previous research efforts have used the strength, time of arrival, or angle of arrival of the pilot signal received at the mobile station from neighboring

base stations and combined it with the known locations of the base stations to solve for the mobile position [5]–[9].

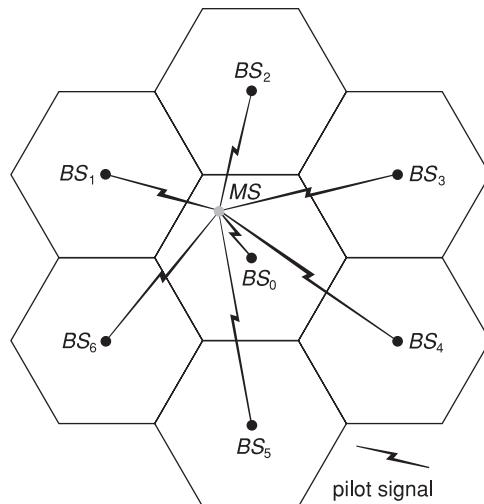
The main problem with the above methods is that the presence of non-line of sight, multiple access interference, and shadowing makes it impossible to obtain an accurate estimate of the mobile location. To overcome this difficulty and in an effort to achieve better results, an adaptive fuzzy approach based on signal strength measurements was proposed in Shen *et al.* [10].

In this work, we propose a mobile location technique using a radial basis function (RBF) network. We propose training an RBF network to estimate the position of a mobile from signal strength measurements. This work is inspired in part by previous successful applications of RBF networks in CDMA systems. Simulations were performed over various propagation environments. These simulations confirmed that the proposed technique gives an accurate estimate when it comes to determining a mobile location.

The remainder of this case study is organized as follows. Section 3 presents the network model of the CDMA cellular network under consideration. Section 4 provides the basic concept behind mobile positioning techniques. The use of an RBF network for mobile location estimation is outlined in section 5. Simulations and results are provided in Section 6. Conclusions are outlined in Section 7.

### 3 Network model

We assume a two-dimensional model of a CDMA cellular system. One such system is the direct sequence code division multiple access (DS-CDMA) system. In this system, the geographic service area over which the mobile units operate is divided into contiguous cells, each of which is served by a single base station as shown in Figure 10.34. In each cell, all the mobiles communicate with the base station using the same allocated frequency range and they are distinguished only by the “spreading codes” they use. In such a system, each base station continuously transmits an unmodulated, direct-sequence spread spectrum signal, termed as the pilot signal. The pilot signal allows a mobile station to obtain the timing of the forward CDMA channel, provides a phase reference for coherent demodulation, and assists in handoff decisions by providing a mean for signal strength comparisons among different neighboring base stations. The mobile station can detect the pilot signal from any base station when the received signal strength is above a certain level which is determined by the transmission power of the pilot signals and the distance between the mobile and the base station. Furthermore, the mobile unit regularly informs the network about the received power levels from all visible base stations in the form of a “pilot strength measurement message.” The message is sent through the forward channel when the mobile is in an idle state, otherwise it will be sent on the reverse channel when the mobile is in traffic mode. This message is sent whenever the received power level of any of the pilot signals from the nearby base stations crosses a certain threshold. The base station may also request the mobile to send this message. The frequency of



**Figure 10.34:** CDMA cellular topology

this message may vary from multiple messages per one second to one message per two seconds, subject to several external factors [4], [11].

## 4 Mobile location techniques

A variety of techniques have been proposed and some of them have been actually used in determining the location of a mobile unit. These technologies may be loosely classified into handset-based techniques, such as the ones based on the global positioning system (GPS), and network-based techniques [3].

Handset-based technologies use the global positioning system and require a GPS receiver in the handset, the cost of which can severely limit their availability to the average consumer.

Network-based techniques benefit from the cellular infrastructure by using the signal being received by a mobile to determine its location. Generally, the mobile signal is received from several base stations with known positions. After reception, some characteristics of that signal are combined with the known positions of the receivers and used to determine the mobile position. This could be the strength of the signal, the time of arrival of the signal, or the angle of arrival of the signal.

### 4.1 Signal strength

This technique is based on measuring the power level of the signal being received by a mobile station from various base stations. This can be used to estimate the distance between the mobile station and each of the individual base stations. Geometrically, this represents circles, centered at the various base stations on which the mobile station must lie. Using at least three base

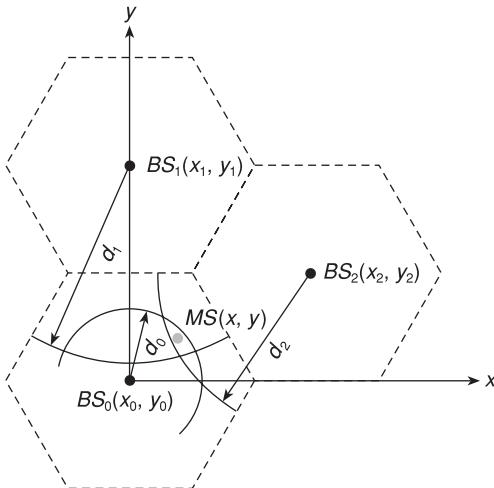


Figure 10.35: Mobile location using signal strength

stations to increase accuracy, the intersection of circles provides the mobile position. With reference to Figure 10.35, the distance between the mobile station and each of the base stations is calculated using the following equations [5]–[6], [10]:

$$\begin{aligned} d_0 &= \sqrt{(x - x_0)^2 + (y - y_0)^2} \\ d_1 &= \sqrt{(x - x_1)^2 + (y - y_1)^2} \\ d_2 &= \sqrt{(x - x_2)^2 + (y - y_2)^2} \end{aligned} \quad (1)$$

The equations above have two unknowns,  $x$  and  $y$ , representing the coordinates of the mobile station.  $[x_0, y_0]$ ,  $[x_1, y_1]$  and  $[x_2, y_2]$  are the coordinates of three neighboring base stations and are assumed to be known. The equations can be solved in many different ways. They can be solved graphically by plotting the three circles they represent. It is also possible to solve the set of equations analytically. One problem arises with this technique when the three circles intersect at more than one point. Taking the average of the possible solutions will increase the estimation error.

## 4.2 Time of arrival

In this technique, the time it takes the mobile signal to reach a number of neighboring base stations is used to estimate the relative location of the mobile to each of the base stations. Each estimated distance confines the location of the mobile to the circumference of a circle centered at the corresponding base station. Using information from three or more base stations, the intersection of the circles provides the mobile position (Figure 10.36). This method is often called the *time of arrival method* and has the disadvantage that it requires the mobile station to act as a transponder in which

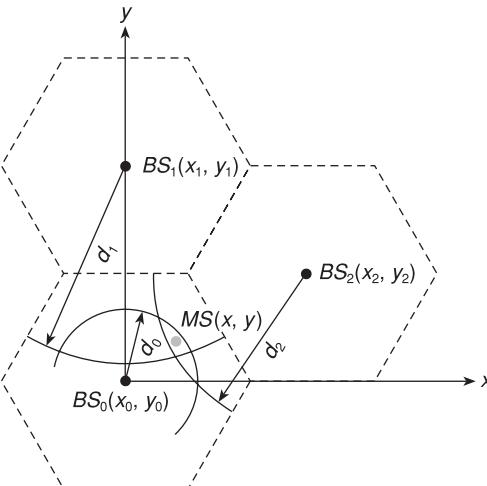


Figure 10.36: Mobile location using time of arrival

processing delays can introduce errors. To overcome this, the time difference rather than absolute measurements has been more commonly used [4], [7].

The distance from the mobile station to each of the base stations can be calculated using the following equations [4]:

$$\begin{aligned} d_0 &= c \cdot t_0 \\ d_1 &= c \cdot t_1 \\ d_2 &= c \cdot t_2 \end{aligned} \tag{2}$$

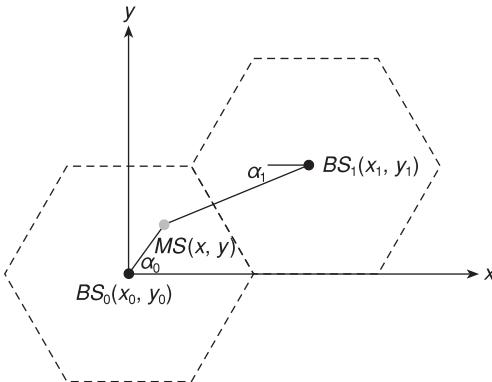
where  $t_0$ ,  $t_1$ , and  $t_2$  are the times it takes for the signals to travel from  $BS_0$ ,  $BS_1$  and  $BS_2$  to the mobile station, respectively and  $c$  is the speed of light. The time difference of arrival algorithm draws two hyperboloids using:

$$\begin{aligned} c_1 &= d_1 - d_0 = c \cdot (t_1 - t_0) = \sqrt{x^2 + (y - y_1)^2} - \sqrt{x^2 + y^2} \\ c_2 &= d_2 - d_0 = c \cdot (t_2 - t_0) = \sqrt{(x - x_2)^2 + (y - y_2)^2} - \sqrt{x^2 + y^2} \end{aligned} \tag{3}$$

The above two equations have two unknowns  $x$  and  $y$  and can be solved in different ways. They can be solved iteratively, graphically, or using Taylor series expansion. It is also possible to solve the equations analytically. A problem with this technique arises when the two hyperboloids don't intersect or when they intersect at more than one point.

#### 4.3 Angle of arrival

The direction of arrival techniques estimate the mobile station location by measuring the angle of arrival of a signal at two different base stations. Two base stations are sufficient for determination of the mobile position.



**Figure 10.37:** Mobile location using angle of arrival

With reference to Figure 10.37, the coordinates of the mobile location can be calculated by the intersection point of the two lines defined by the equations [8], [9]:

$$\begin{aligned} y &= \tan(\alpha_0)x \\ y - y_1 &= (x - x_1)\tan(\alpha_1) \end{aligned} \tag{4}$$

$\alpha_0$  and  $\alpha_1$  are the angle of arrivals of the mobile signal received at base station  $BS_0$  and  $BS_1$  respectively. Solving the above equations for  $x$  and  $y$  yields the coordinates of the mobile location:

$$x = \frac{y_1 - x_1 \tan(\alpha_1)}{\tan(\alpha_0) - \tan(\alpha_1)} \tag{5}$$

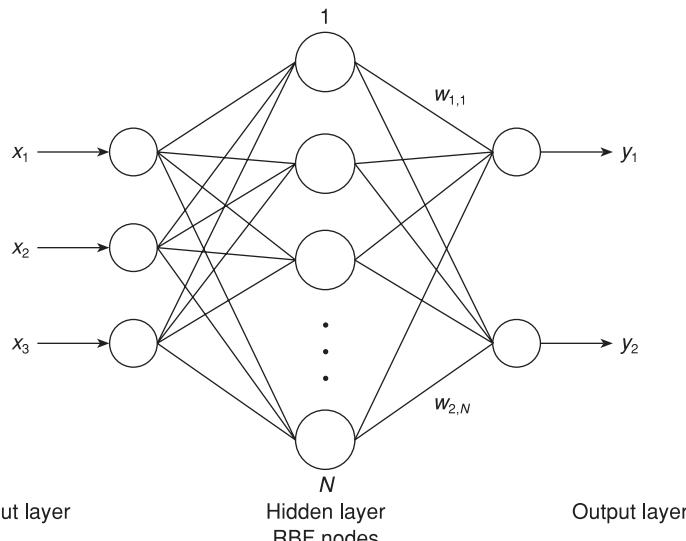
and

$$y = \frac{y_1 \tan(\alpha_0) - x_1 \tan(\alpha_0) \tan(\alpha_1)}{\tan(\alpha_1) - \tan(\alpha_0)} \tag{6}$$

## 5 Solution with RBF network

Common to all the techniques we have discussed in the previous section is the inherent nonlinearity in estimating the mobile location from the signal characteristics. The techniques work well for an ideal propagation environment where no noise is present, the measurements are accurate, and the signal propagation follows a line of sight.

RBF networks offer a good tool for modeling such nonlinearities. In addition to being universal approximators, their learning speed is relatively fast. We use here an RBF network to estimate the location of a mobile from



**Figure 10.38:** RBF neural network

the measured signal strength received from three different base stations. Any of the signal characteristics, namely strength, time of arrival, or angle of arrival can be used.

In order to use the RBF network for estimating the location of a mobile, we set the inputs of the network equal to the three strongest signals received by the mobile from all visible base stations. Let these values be represented by the input vector  $X = [x_1, x_2, x_3]^T$ . The output vector  $Y = [y_1, y_2]^T$  is set to be the coordinates representing the location of the mobile unit to be positioned. The structure of the RBF network is shown in Figure 10.38. The output of the  $i$ -th RBF neuron is:

$$\phi_i(X) = \phi_i\left(\frac{\|X - C_i\|}{\sigma_i}\right), \quad i = 1, \dots, N \quad (7)$$

where  $X$  is the three-dimensional input vector,  $C_i$  is a vector with the same dimension as  $X$ ,  $N$  is the number of hidden nodes. The function  $\phi_i(\cdot)$  is typically chosen to be Gaussian:

$$\phi_i(X) = \exp\left(-\frac{\|X - C_i\|^2}{\sigma_i^2}\right) \quad (8)$$

The  $j$ -th output  $y_j(X)$  of the RBF neural network is calculated by the equation:

$$y_j(X) = \sum_{i=1}^N w_{i,j} \times \phi_i(X); \quad j = 1, 2 \quad (9)$$

Using training data which consist of a set of input/output vectors, the network weights are solved using the least square method.

## 6 Results

For evaluating the performance of this estimation technique, we consider the cell topology presented in Section 3 and shown in Figure 10.34. We assume that the mobile is located in the central cell  $BS_0$ . The mobile is then moved within the central cell  $BS_0$ . Due to the symmetric nature of the cell arrangement, the area covered by  $BS_0$  will be representative of the areas covered by the neighboring cells.

In the first phase of the simulation, radiolocation data are used to train the RBF network. The radiolocation data consist of a set of pilot signal power levels received at different mobile locations and corresponding mobile positions. These data could be obtained from the Mobile Switching Center of a network operator. Another possibility is to use a cellular network simulator.

For our simulation we consider a typical path-loss model, in which the signal strength attenuates according to an inverse power law such that the degradation at a distance  $d$  is  $d^{-k}$  [11]. The exponent  $k$  is dependant on the propagation environment and varies between 2 and 6. Furthermore, the signal is multiplied by a lognormal random variable, representing shadow fading. Experiments suggest values ranging from 2 to 6 db for the standard deviation of the random variable modeling shadow fading [11]. When the mobile is located at a distance  $d$  from the base station, the average power of the received signal at the mobile can then be calculated as:

$$a(d) = d^{-k} 10^{G(0, \sigma^2)/10} \quad (10)$$

With the calculated strength of the pilot signal from different base stations, the three largest signal strengths are chosen for the estimation of the location of the mobile station. We considered 5000 randomly generated training data using the above propagation model to train the RBF network.

Once the RBF network is trained to estimate the location of a mobile station given the power levels received from the neighboring base stations, the testing data are used to assess the accuracy with which the position is estimated. In the testing phase, we simulated 200 mobile stations moving through random trajectories. For the purpose of comparing our result with those obtained in [10], random mobile trajectories similar to those used in [10] were generated in Matlab. The trajectories are characterized by the following:

- (1) The initial location of each mobile is randomly chosen according to a uniform distribution in the central cell.
- (2) The initial and subsequent velocity of each mobile is a constant randomly chosen according to a uniform distribution in the interval [10, 30] meters per second.
- (3) The initial and subsequent direction of motion of each mobile is randomly chosen according to a uniform distribution in the interval  $[0, 2\pi]$ .
- (4) The location information is updated each second.

**Table 10.9:** Root mean square estimation error given  $k = 4$ 

$\sigma(\text{dB})$	Shen <i>et al.</i> $\mu_{\text{RMSE}}(\text{m})$	$\mu_{\text{RMSE}}(\text{m})$	Proposed technique $\sigma_{\text{RMSE}}(\text{m})$
2	104.474	76.4958	56.2445
3	140.504	98.0217	82.4870
4	185.780	138.9156	116.2045
5	249.476	154.3670	126.4734
6	314.024	178.4388	138.4837

We use the root mean square error as the performance index for comparing the true locations and their obtained estimates:

$$\sqrt{\frac{1}{N} \sum_{n=1}^N (\hat{x}_n - x_n)^2 + (\hat{y}_n - y_n)^2} \quad (11)$$

As the mobile station moves, the distance to each of the base stations is calculated using equation 1 and the power levels are estimated and used as inputs to the RBF network. The output of the network gives the estimated location.

The set of all the estimated locations are joined to form the estimated trajectory. The sample mean and variance of the root mean square error statistics were computed using 200 independently generated sample trajectories for different propagation environments (i.e., different values of  $k$  and  $\sigma$ ). Table 10.9 gives the mean square estimation errors and their standard deviations for various values of the noise power with the attenuation constant set to  $k = 4$ . As seen from the results as the fading power, represented by the variance  $\sigma^2$ , decreases, the randomness in the signal decreases, and better estimates are obtained. Table 10.10 gives the mean square estimation errors for different values of  $k$  with  $\sigma = 2$  dB. It was observed that in higher attenuation environments, the estimation accuracy is lower. This can be interpreted by the fact that the higher the attenuation, the lower the signal to noise ratio, and hence the higher the uncertainty in the signal.

In our simulation, we compared the fuzzy inference system used in [10] and our proposed algorithm in different propagation environments. The tables provide a quantitative comparison of the relative merits of the two techniques, and show the relatively better performance of the technique proposed in terms of obtaining a lower root mean square error for various

**Table 10.10:** Root mean square estimation error given  $\sigma = 2$  dB

$\sigma(\text{dB})$	Shen <i>et al.</i> $\mu_{\text{RMSE}}(\text{m})$	$\mu_{\text{RMSE}}(\text{m})$	Proposed technique $\sigma_{\text{RMSE}}(\text{m})$
2	190.525	126.9146	101.8120
4	104.474	76.4958	56.2445
6	71.443	64.0767	46.0306

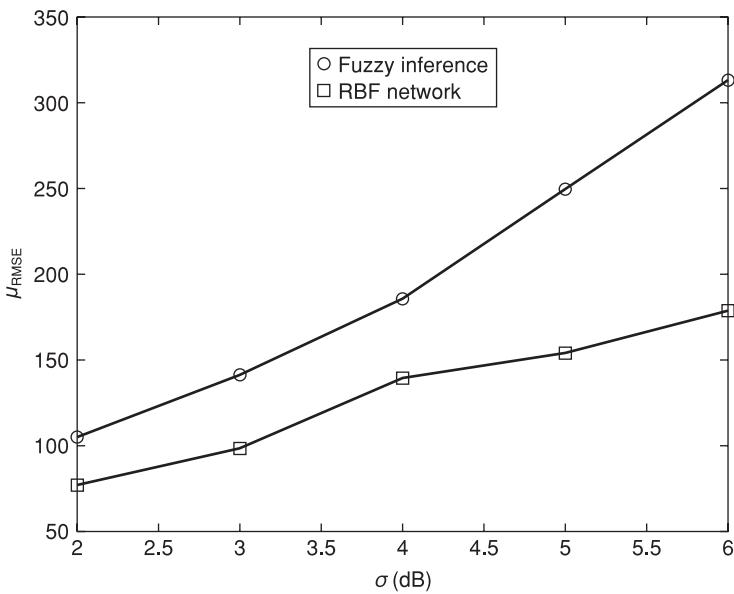


Figure 10.39: Root mean square estimation error with  $k = 4$

propagation environments. The standard deviation was calculated and tabulated to give a confidence level of how far the estimates may deviate from the mean estimates. Figure 10.39 depicts the results in graphical form. In Figure 10.40, a sample of the generated mobile station movement trajectories and its corresponding estimate are shown.

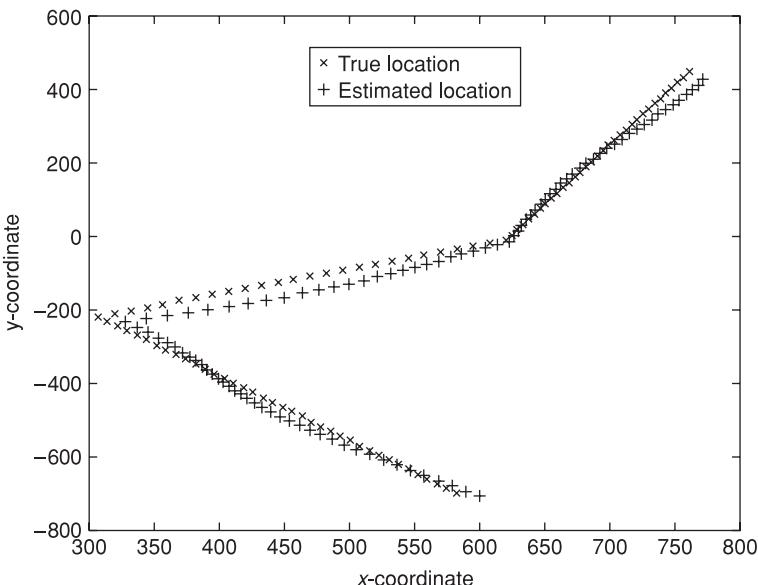


Figure 10.40: Sample generated trajectory and its corresponding estimate

## 7 Concluding remarks

This case study presented the use of an RBF network to determine the location of a mobile station based on the signal power received by several base stations. Simulations in different environments have shown the merit of the technique. The process can be generalized in order to locate the mobile in any environment using the same approach.

### References

1. FCC, "Guidelines for Testing and Verifying the Accuracy of Wireless E911 Location Systems," *OET BULLITEN No. 71*, April 12, 2000.
2. IST-1999-14093 LOCUS Deliverable D4, "Recommendations," *Information Society Technologies*, September 2001.
3. Caffery, J.J., and Stuber, G.L. (1998) "Subscriber location in CDMA cellular networks," *IEEE Transactions on Vehicular Technology*, vol. 47, no. 2, pp. 406–16.
4. Sunay, M.O., and Tekin, I. (1999) "Mobile location tracking in DS CDMA networks using forward link time difference of arrival and its application to zone-based billing," in *Proceedings of the Global Telecommunications Conference, GLOBECOM 99*, pp. 143–147, December.
5. Hata, M., and Nagatsu, T. (1980) "Mobile location using signal strength measurements in a cellular system," *IEEE Transactions on Vehicular Technology*, vol. 29, pp. 245–252.
6. Messier, G., Fattouche, M., and Petersen, B.R. (1998) "Locating an IS-95 mobile using its signal," *The Tenth International Conference on Wireless Communications (Wireless 98)*, vol. 11, (Calgary, AB, Canada), pp. 562–574, July 6–8.
7. Ho, K.C., and Chan, Y.T. (1997) "Geolocation of a known altitude object from TDOA and FDOA measurements," *Transactions on Aerospace and Electronic Systems*, vol. 33, no. 3, pp. 770–783.
8. Wong, K.T. (1998) "Adaptive geolocation and blind beamforming for wide-band fast frequency-hop signals of unknown gop sequences and unknown arrival angles using an electromagnetic vector sensor," *IEEE International Conference on Communications*, pp. 758–762.
9. Klukas, R., and Fattouche, M. (1998) "Line-of-sight angle of arrival estimation in the outdoor multipath environment," *IEEE Transactions on Vehicular Technology*, vol. 47, no. 1, pp. 342–351.
10. Shen, X., Mark, J.W., and Ye, J. (2002) "Mobile location estimation in CDMA cellular networks by using fuzzy logic," *Wireless Personal Communications*, vol. 22, no. 1, pp. 57–70.
11. Rappaport, T.S. (2001) *Wireless Communications: Principles and Practice*, Prentice Hall, Englewood Cliffs.

## Case study 5

### Learning-based resource optimization in ATM networks

© 2003 IEEE. Reprinted, with permission, from “Learning Based Resource Optimization in Asynchronous Transfer Mode (ATM) Networks,” S. Alsharhan, F. Karray and W. Gueaieb, *IEEE Transactions on Systems, Man and Cybernetics, Part B*, vol. 33, no. 1, February 2003, pp. 122–32.

## 1 Introduction

This case study tackles the issue of bandwidth allocation in ATM networks using recently developed tools of computational intelligence. Efficient bandwidth allocation technique implies effective resources utilization of the network. The fluid flow model has been used effectively among other conventional techniques to estimate the bandwidth for a set of connections. However, such methods have been proven to be inefficient at times in coping with varying and conflicting bandwidth requirements of the different services in ATM networks. This inefficiency is due to the computational complexity of the model. To overcome this difficulty, many approximation-based solutions, such as the fluid flow approximation technique, were introduced. Although such solutions are simple in terms of computational complexity, they nevertheless suffer from potential inaccuracies in estimating the required bandwidth. Soft computing-based bandwidth controllers, such as neural networks and neuro-fuzzy-based controllers, have been shown to effectively solve an indeterminate nonlinear input–output relation by learning from examples. Applying these techniques to the bandwidth allocation problem in ATM network yields a flexible control mechanism that offers a fundamental trade-off for the accuracy–simplicity dilemma.

## 2 Background

Asynchronous transfer mode (ATM) networks have become in recent years among the promising technologies for supporting broadband multimedia services [1]. It has been widely recognized that ATM technology has the necessary capability to handle the diversity of traffic that is foreseen for broadband integrated services digital networks (B-ISDN). For example, the packets in ATM networks have a fixed size of 53 bytes. In addition, a virtual path concept has been introduced in ATM networks [2]. This concept reduces the network operating control and the connection setup time, and simplifies the connection admission control (CAC). ATM has been designed to support a mixture of multimedia traffic (e.g., audio, video, data, images) with different

traffic parameters, such as peak bit rate, burst time, cell delay transfer delay, and different Quality of Services (QoS) requirements [3]. The ATM layer supports several service classes that impose different QoS, such as constant bit rate (CBR), real-time variable bit rate (rt VBR), non-real-time variable bit rate (nrt VBR), available bit rate (ABR), and unspecified bit rate (UBR). However, ATM networks still represent some challenging problems in terms of properly utilizing the network resources (e.g., the bandwidth) and in properly guaranteeing QoS for all the different traffic classes without suffering from conflicts. Moreover and as a technology that supports multimedia traffic, ATM networks suffer from unpredictable fluctuations and burstiness of traffic.

Among the main tasks of any bandwidth allocation technique in ATM networks is to protect the resources against overload [4], and to maximize the utilization of the network links' capacity. Conventional methods that tackle the bandwidth allocation problem suffer from serious limitations. They cannot support all of the ATM features because they are designed for pre-determined and well-specified situations. Furthermore, while some of these techniques are accurate [5], they are nevertheless based on complex mathematical models. As such, deriving a precise model for the system could become a difficult undertaking in real-time environments. Other proposed bandwidth allocation techniques [6] are light in terms of computational complexity and are suited well to the real-time requirements. However, they are based on a number of approximations that may lead to an inaccurate bandwidth estimation (e.g., overestimation). In ATM traffic management, one may notice that the connection admission control (CAC) procedure and bandwidth allocation controller have to tackle the accuracy–simplicity dilemma.

On the other hand, techniques based on tools of soft computing may not have such a limitation as they could possibly operate over a continuous space [7] of operating conditions. In fact and as described earlier in this textbook, computational intelligence tools have been credited in a number of applications with providing satisfactory results in the face of relatively large amounts of unpredictability, fluctuation, and in the absence of a precise model of the system.

In this case study, two soft computing-based techniques (i.e., neural networks and neuro-fuzzy systems) are proposed to tackle the bandwidth allocation procedure in ATM networks. The remainder of this case study is organized as follows: in section 3, an overview on the conventional model of bandwidth allocation is provided. Particular focus is put on the fluid flow model and fluid flow approximation model. In section 4, we introduce a number of soft computing-based techniques along with their implementation for bandwidth allocation in ATM networks. In section 5, simulation results are reported and discussed. Section 6 presents some concluding remarks.

### 3 Overview of conventional bandwidth allocation techniques

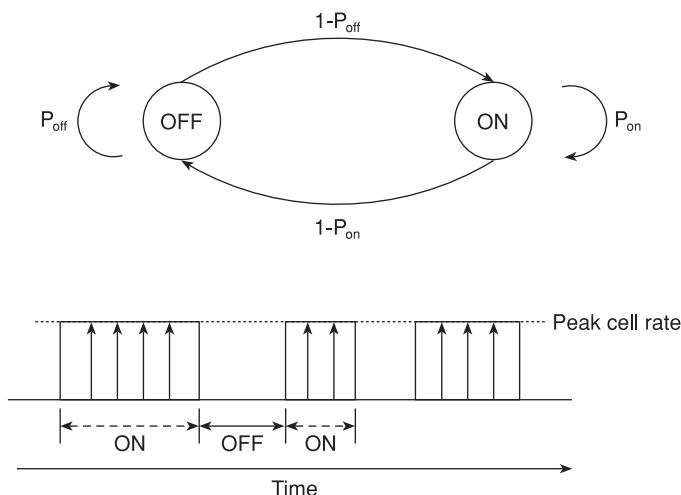
In ATM multiplexing there are few approaches to handle the access multiplexer [8]. Among these approaches, the fluid flow model is considered the

most accurate, and it has been popular in applications involving communication traffic modeling and characterization [5]. The exact fluid flow can achieve the best utilization of the bandwidth. However, this scheme cannot be implemented in real-time environments due to the analysis complexity and the bursty nature of the traffic [9]. Hence, simplified model schemes have been proposed in order to achieve a proper trade-off of the performance-complexity dilemma. Effective bandwidth is an example of such models. It represents the minimum amount of bandwidth that achieves the QoS requirements for a set of connections multiplexed on a limited capacity link. The effective bandwidth scheme aims at a higher utilization of the link. Another objective is to maximize the number of accepted calls since the effective bandwidth is connected with the admission connection control procedure. Because of its attractive simplicity, many effective bandwidth schemes have been proposed [10], [11], [6], [12], [13], [14], where for every connection, an effective bandwidth is assigned according to:

$$R_a \leq c \leq R_p \quad (1)$$

where  $R_a$  is the average cell rate,  $R_p$  represents the peak cell rate, and  $c$  is the effective bandwidth.

In this case study, we emphasize the exact fluid flow model and the effective bandwidth based on its approximation to show that soft computing techniques effectively tackle such complex problems in ATM environments. We also study the ON-OFF sources seeking for simplicity in modeling bursty sources. An ON-OFF source is illustrated in Figure 10.41 along with its traffic model. During the ON (active) period, the source generates cells at the peak cell rate ( $R_p$ ), while during the OFF (idle) period no cells are generated. In such a two-state Markov source, the ON and OFF periods are exponentially distributed. The rest of this section tackles the exact fluid flow model [15] and its approximation as proposed by Guerin *et al.* [6].



**Figure 10.41:** ON-OFF source

### 3.1 The exact solution: fluid flow model

The fluid flow model deals with the sources as continuous fluid streams. It assumes that each active source sends information cells uniformly and the server tackles them uniformly. The fluid flow approach was pioneered by Anick, Mitra, and Sondhi [15]. To illustrate this approach, consider a statistical multiplexing system that consists of  $N$  ON-OFF fluid sources (not necessarily homogeneous). The sources are statistically independent Markov-modulated where the ON (spurt) and OFF (idle) periods are exponentially distributed. When the arrival rate exceeds the link capacity, the incoming cells are stored in an infinite capacity buffer. The average ON period ( $1/\alpha$ ) is selected to be the “unit” of time. Using this unit, the average OFF period is  $(1/\lambda)$ , where  $\lambda$  is the rate of transition out of the OFF state. Similarly, the unit of information is chosen to be the amount generated by the ON source during a unit of time. Using these notations, the receiving rate at the buffer is  $i$  if there are  $i$  instant ON sources, and for a non-empty buffer the rate of change of the buffer content is  $i - c$ , where  $c$  is the required link capacity. For a stable system, the allocated bandwidth must satisfy the following stability condition:

$$\frac{R_p N \lambda}{(1 + \lambda)} < c \quad (2)$$

where  $R_p$  is the peak cell rate of the sources. To find the distribution of the buffer occupancy, let's define  $F_i(t, x)$ ,  $0 \leq i \leq N$  to be the buffer cumulative probability distribution at time  $t$ , with  $x$  being a continuous random variable denoting the buffer occupancy at the time  $t$ . In other words,  $F_i(t, x)$  represents the probability of the buffer occupancy which does not exceed  $x$  at a time where there are  $i$  sources in the active state (ON state). At the next time interval  $\Delta t$ , either a source goes from the OFF state to the ON state or vice versa, with probabilities  $[N - (i - 1)]\lambda\Delta t$  and  $(i + 1)\alpha\Delta t$ , respectively. Hence, the probability of no change becomes  $1 - [(N - i)\lambda + i\alpha]\Delta t$ . Using these probabilities, the buffer cumulative probability distribution during the next time interval can be defined as follows:

$$F_i(t + \Delta t, x) = [N - (i - 1)]\lambda\Delta t F_{i-1}(t, x) + (i + 1)\alpha\Delta t F_{i+1}(t, x) + \{1 - [(N - i)\lambda + i\alpha]\Delta t\}F_i[t, x - (i - c)\alpha\Delta t] + O(\Delta t^2) \quad (3)$$

where  $O(\Delta t^2)$  is the probability of compound events. Now, passing the limit  $\Delta t \rightarrow 0$ , (3) becomes expressed as:

$$\frac{\partial F_i(x, t)}{\partial t} - (i - c)\alpha \frac{\partial F_i}{\partial x}(t, x) = [N - (i - 1)]\lambda F_{i-1}(t, x) + (i + 1)\alpha F_{i+1}(t, x) - [(N - i)\lambda + i\alpha]F_i(t, x) \quad (4)$$

Since we are mainly interested in the equilibrium probability, equation 4 can be re-written under the steady-state condition  $\left(\frac{\partial F_i(t, x)}{\partial t} = 0 \text{ and } F_i(t, x) \rightarrow F_i(x)\right)$  as follows:

$$(i - c)\alpha \frac{dF_i(x)}{dx} = [N - (i - 1)]\lambda F_{i-1}(x) - [(N - i)\lambda + i\alpha]F_i(x) + (i + 1)\alpha F_{i+1}(x) \quad (5)$$

To solve the above set of equations, let's define an  $(N + 1)$  element vector  $F(x)$  as:

$$F(x) = [F_0(x), F_1(x), \dots, F_N(x)] \quad (6)$$

Equation (5) can then be rewritten in the following compact form:

$$\frac{dF(x)}{dx} D = F(x) M \quad (7)$$

where  $D$  is an  $(N + 1) \times (N + 1)$  diagonal matrix given by:

$$D = \text{diag}[-c\alpha, (1 - c)\alpha, \dots, (N - c)\alpha] \quad (8)$$

and  $M$  is an  $(N + 1) \times (N + 1)$  matrix defined as:

$$M = \begin{bmatrix} -N\lambda & N\lambda & 0 & \dots & \dots \\ \alpha & -[\alpha + (N - 1)\lambda] & (N - 1)\lambda & \dots & \dots \\ 0 & 2\alpha & -(N - 2)\lambda - 2\alpha & \dots & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & 2\lambda & 0 \\ \vdots & \vdots & \vdots & -(N - 1)\alpha - \lambda & \lambda \\ 0 & \vdots & \vdots & N\alpha & -N\alpha \end{bmatrix} \quad (9)$$

Equation (7) becomes expressed as follows:

$$\frac{dF(x)}{dx} = F(x) M' \quad (10)$$

where  $M' = MD^{-1}$ .

The equilibrium distribution of the buffer contents of an  $N$ -state Markov chain takes the form:

$$F(x) = \sum_{i=1}^N a_i \Phi_i e^{z_i x} \quad (11)$$

where  $z_i$  and  $\Phi_i$  are the generalized eigenvalues and eigenvectors associated with the solution of differential equations satisfied by the stationary probability of the system. The coefficients  $a_i$ 's are determined using boundary condition [15]. In the case of  $N$  bursty Markov sources, the probability of buffer overflow (QoS) is defined as follows:

$$G(x) = \beta e^{-rx} \quad (12)$$

with

$$\beta = \rho^N \prod_{i=1}^{N-[c/R_p]-1} \frac{z_i}{(z_i + r)} \quad (13)$$

$$\text{where } r = \frac{(1+\lambda)(1-\rho)}{1-c/NR_p} \text{ and } \rho = \frac{N\lambda}{c(1+\lambda)} < 1.$$

Let  $\varepsilon$  be the overflow probability which is equal to the cell loss probability. The smallest amount  $\hat{c}$  of  $c$  that guarantees the QoS (i.e., overflow probability is always less than  $\varepsilon$ ) is called the effective bandwidth. The problem here is to find  $\hat{c}$  from (12). However, one can notice that  $\beta$  is a complex expression and involves both the largest eigenvalue and other negative eigenvalues. The solution can be tackled using iterative numerical methods which are difficult to implement in real-time environments. Due to this complexity, the fluid flow model is not suited for real-time traffic, and hence, other solutions are sought to tackle in real time the bandwidth allocation problem. Many approximation-based solutions have been proposed in [6], [13], [16]. Among the most popular solutions is the fluid flow approximation introduced in [6].

### 3.2 Fluid flow approximation

Solving for the exact bandwidth using the above fluid flow model is a very complex problem, especially for the case of heterogeneous sources. Guerin *et al.* [6] introduced an approximation that provides the upper bound of the effective bandwidth. This method approximates the  $\beta$  expression by 1. Hence, for a single two-state source, the effective bandwidth is taken to be:

$$\hat{c} = \frac{\alpha b(1-\rho)R_p - x}{2\alpha b(1-\rho)} + \frac{\sqrt{[\alpha b(1-\rho)R_p - x]^2 + 4x\alpha b\rho(1-\rho)R_p}}{2\alpha b(1-\rho)} \quad (14)$$

where  $\alpha = \ln\left(\frac{1}{\varepsilon}\right)$ . The effective bandwidth for  $N$  multiplexed sources is given by:

$$C = \sum_{i=1}^N \hat{c} \quad (15)$$

Although the fluid flow approximation method has the merit of computational simplicity, it has its own shortcomings. In fact, there is no consideration of any multiplexing aspect. It assumes that cells arrive at the buffer uniformly and are removed similarly. In some situations, fluid flow approximation leads to inaccurate results [16], [17]. Fluid flow approximation produces an upper bound of the effective bandwidth that always overestimates the amount of bandwidth required by a set of sources. This can be evidently seen when the utilization factor ( $\rho$ ) is low. Hence, one can conclude

that the computational simplification in the fluid flow approximation results in reduction of accuracy.

## 4 Soft computing techniques

The dynamic nature of the multimedia traffic along with the uncertainty found in the relations linking traffic parameters in ATM networks is an obstacle for attaining an efficient bandwidth allocation controller. Neural networks and neuro-fuzzy systems have attractive features that motivate their use in bandwidth allocation. They have the capability of learning from sets of data that describe the traffic parameters and are able to estimate the correct bandwidth. Also, their parallelism and adaptability help them overcome the efficiency restriction of the conventional methods. Beside these merits, the neuro-fuzzy-based bandwidth allocation controller has the following advantages [7], [18], [19], [20]:

- Handles any kind of information (numeric, linguistic, logical).
- Resolves conflicts by aggregation.
- Self-learning, self-organizing and self-tuning capabilities.
- No need of prior knowledge of data relationships.
- Fast computation using fuzzy number operations.

For these merits, soft computing techniques are considered as a potential candidate to solve the accuracy–simplicity dilemma that faces the conventional approaches. Soft computing techniques such as those based on fuzzy logic, neural networks, and neuro-fuzzy systems have been applied to handle many aspects in ATM networks. A comprehensive survey on the applications of soft computing techniques in ATM networks can be found in [21], [22].

### 4.1 The model

The proposed model considers soft computing-based (neuro-fuzzy and neural networks) bandwidth allocation controllers in ATM networks. The architecture of the controller is designed so as to accommodate a wide variety of Quality of Service (QoS) requirements. The QoS criterion is chosen to be the cell loss probability. The system model of the bandwidth allocation controller is depicted in Figure 10.42.

In both neural network-based and neuro-fuzzy-based bandwidth allocation controllers the inputs are the cell loss probability (QoS), the rate of transition out of the OFF state,  $\lambda$ , and the number of sources of the identical class,  $N$ . The output of the controllers is taken to be the normalized effective bandwidth with respect to the peak cell rate ( $R_p$ ). Thus, the proposed controllers can estimate the bandwidth for heterogeneous classes based on the diversity of cell loss probability. In the proposed model the incoming traffic is scattered into  $m$  classes of homogeneous traffic. Each class consists of  $N$  sources

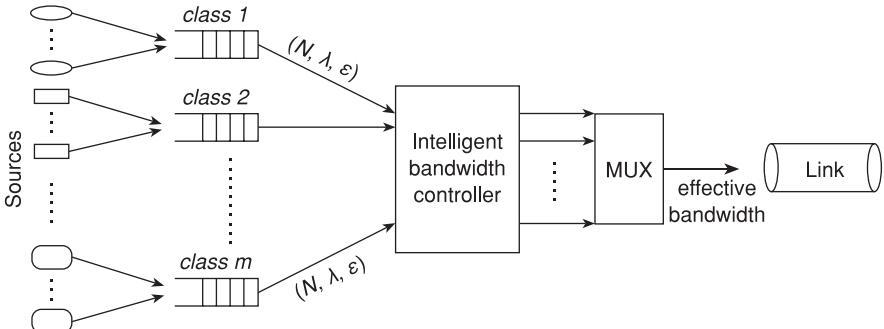


Figure 10.42: The system model

with the same QoS (cell loss probability), and  $\lambda$ . The peak cell rate ( $R_p$ ) is normalized to one. The intelligent bandwidth controllers estimate the required bandwidth for each class, where the statistical multiplexing is taken into consideration. The total bandwidth of all classes is taken to be the sum of all classes' bandwidth, given as:

$$C = \sum_{j=1}^m \hat{c}_j \quad (16)$$

When the switch receives a request for a new connection, it categorizes it into a bit-rate class according to the declared parameters [23]. The intelligent bandwidth controller estimates the required bandwidth for this connection and sends the action to the connection admission controller which decides on whether to accept or reject the connection according to the bandwidth availability.

## 4.2 Neuro-fuzzy-based controller

Neuro-fuzzy systems have been used as effective control systems that integrate the best features of fuzzy logic systems with those of ANNs. They are based on the optimization of the membership functions of a fuzzy logic inference engine system. The membership functions are modified through a learning process as fine-tuning. Following the training and learning stages, a well-tuned neuro-fuzzy controller usually results in the generation of a faster and more accurate output than that obtained by a conventional neural network [18]. The process of forming the membership functions and generating the rules is done based on a least square tuning algorithm which considers the actual output of the training data with the calculated output from the fuzzy system. The tuning process is applied off line and aims at minimizing the error between the desired and actual outputs. Once trained, the neuro-fuzzy controller operates in a very similar fashion to that of a neural network-based controller.

The neuro-fuzzy bandwidth controller, which is a strongly coupled system, has the same structure as a traditional fuzzy system except that each task

**Table 10.11:** The linguistic variables associated with the membership functions of each input

Input	Linguistic variables			
$N$	small	Lmedium	Hmedium	large
$\lambda$	short	medium	long	
$\varepsilon$	low	medium	high	

of the fuzzy inference system is performed by a layer of hidden neurons as described in Chapter 7 and as reported in [20]:

*Layer 1* – This layer consists of three input nodes where each node stores the membership function of a linguistic value corresponding to one of the system's input variables ( $N$ ,  $\lambda$ ,  $\varepsilon$ ). The translation of the input into linguistic variables is achieved by defining a generalized bell function. Thus, the output of each node in this layer can be defined as

$$O_i^{(1)} = \mu_{A_i^{(1)}}(x_i), \quad i = 1, \dots, 3, \quad j = 1, \dots, J, \quad (17)$$

where  $x$  is the input to node  $i$  (i.e.,  $N$ ,  $\lambda$ , or  $\varepsilon$ ),  $J$  is the number of the linguistic labels,  $A_j$ , associated with this node, and  $\mu_{A_i}$  is the membership function for  $A_i$  given as the generalized bell function:  $\mu_{A_i} = \frac{1}{1 + \left| \frac{x - c_i}{a_i} \right|^{2b}}$ , where

$\{a_i, b_i, c_i\}$  is the parameter set associated with the function. Table 10.11 illustrates the linguistic variables that are related to the membership functions of each input.

*Layer 2* – Each node in this layer calculates the firing strength of one rule  $R^{(l)}$  by performing a T-norm operation (we choose the AND operator) on all the membership degrees in the antecedent of the rule. The firing strength  $w_l$  of rule  $R^{(l)}$  is computed as

$$w_l = O_l^{(2)} = \prod_{i=1}^3 \mu_{A_i^{(l)}}(x_i), \quad l = 1, \dots, L. \quad (18)$$

where  $L$  denotes the total number of the fuzzy *if-then* rules.

*Layer 3* – For each rule,  $R^{(l)}$ , there corresponds a node that calculates its weighted firing strength  $\bar{w}_l$ , i.e.;

$$\bar{w}_l = O_l^{(3)} = \frac{w_l}{\sum_{l=1}^L w_l}, \quad l = 1, \dots, L. \quad (19)$$

*Layer 4* – Each node of this layer is connected to all the input nodes and with exactly one node in the third layer to compute the output rule  $R(l)$ .

$$O_l^{(4)} = \bar{w}_l f_l = \bar{w}_l (p_i N + q_i \lambda + k_i \varepsilon + z_i) \quad (20)$$

**Table 10.12:** Design parameters of the neuro-fuzzy-based controller

Parameter	Value
Number of inputs	Three ( $N, \lambda, \varepsilon$ )
Membership functions	( $N = 4, \lambda = 3, \varepsilon = 3$ )
Number of outputs	One ( $BW$ )
Number of layers	Five (ANFIS-based controller)
Learning algorithm	Hybrid learning
Training set size	650 input–output vectors
Epochs number	3000

where  $\bar{w}_l$  is the output of the  $l$ th node of layer 3, and  $\{p_i, q_i, k_i, z_i\}$  is a parameter set associated with the coefficients of the output function in Sugeno-type fuzzy rules since ANFIS adopts the Sugeno-type fuzzy engine.

*Layer 5* – This layer is composed of one node that computes the final output as the summation of all incoming signals.

$$O_l^{(5)} = \sum_{l=1}^L \bar{w}_l f_l. \quad (21)$$

In order to achieve faster parameter identification, we made use of the hybrid learning algorithm [20] that combines the steepest descent and the least-squares estimator (LSE) to update the parameters in the network. The hybrid learning trains the system off line where the gradient descent is used to learn the antecedent parameters, while the least-squares method is used to learn the consequent parameters. Table 10.12 summarizes the design parameters of the neuro-fuzzy-based bandwidth allocation controller while Figure 10.43 depicts the membership function for each input before and after training.

### 4.3 Artificial neural network-based controller

Artificial neural networks (ANNs), as reported earlier in the book, represent an important class of numerical learning tools. Their nonlinear character, parallel distributed processing, and optimization and learning capabilities make them highly useful for numerical modeling and control, particularly for systems for which only little is known about their dynamics and operating environment.

The neural network-based bandwidth controller proposed here is of the multilayer perceptron (MLP) type with the Levenberg–Marquardt (LM) algorithm used as its learning algorithm. The LM algorithm has been proven to be one of the fastest learning algorithms. The proposed controllers are trained with a set of data extracted from the fluid flow model [15]. The input–output vectors consist of  $[N, \lambda, \varepsilon]$  as an input vector and the normalized effective bandwidth as output. The generation of training data set is based on simulating a stream type of different bit-rate traffic (classes) of different QoS.

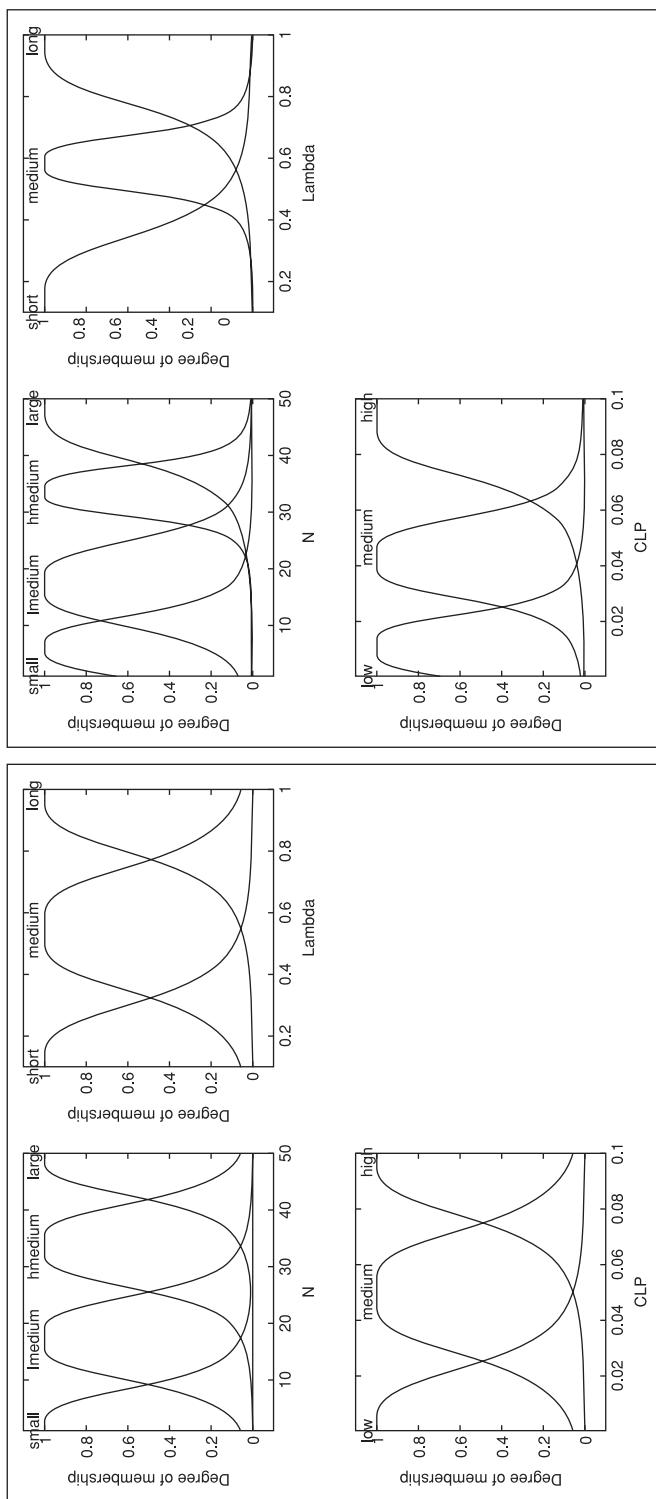


Figure 10.43: Membership functions of the inputs before (left) and after training (right)

**Table 10.13:** Design parameters of the neural network-based controller

Parameter	Value
Number of inputs	Three ( $N, \lambda, \varepsilon$ )
Number of outputs	One ( $BW$ )
Number of layers	Three
Hidden layers	One
Hidden layer nodes	22
Learning algorithm	Levenberg–Marquardt (LM)
Learning rate	0.2
Training set size	650 input–output vectors
Epochs number	3000

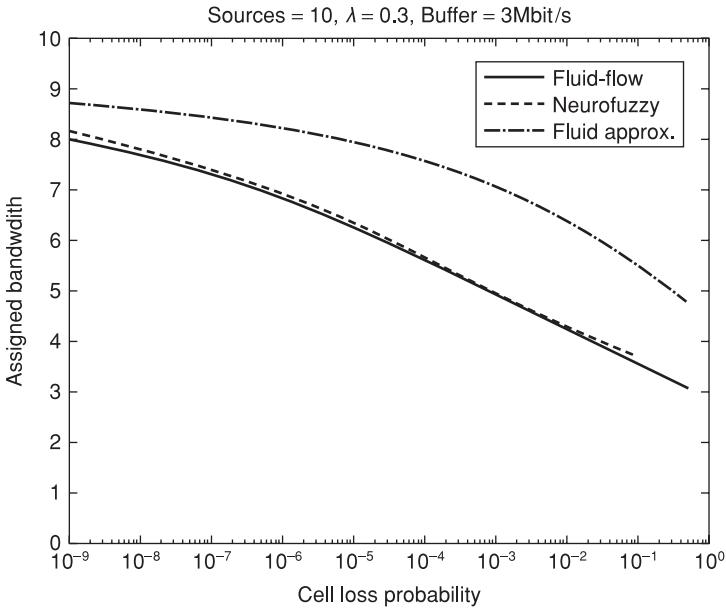
The number of sources in each class is limited to 100 sources, and  $\lambda$  is restricted to the values of  $\{0.01, 0.1, \dots, 1\}$ . The values of the cell loss probability (QoS) are in the range of  $10^{-1}$  to  $10^{-9}$ . Table 10.13 presents the design parameters of the neural-network-based controller.

The neuro-fuzzy-based and neural network-based bandwidth allocation controllers have been used to allocate the bandwidth for traffic with variant QoS. Once they are trained they can be used in a real-time environment to allocate the required bandwidth without the need for an explicit mathematical model. In contrast to sequential algorithms, an inherent parallelism feature in such systems reduces the processing time of the traffic. It also enables the proposed controllers to effectively predict the fluctuation in the traffic with a small reaction time.

## 5 Simulation results

To determine the accuracy of the proposed intelligent bandwidth controllers, their output is compared to the output of the fluid flow approximation, proposed by Guerin *et al.* [6]. In this section a number of numerical examples are provided and demonstrate the performance of the neuro-fuzzy-based and neural network-based bandwidth allocation controllers. The numerical results are obtained from simulating the intelligent controllers, fluid flow model, and the fluid flow approximation models. For these examples, the buffer size is taken to be 3 Mbit/s and the burst period takes the value of 1. In this section, the results of several experiment sets are explained in order to compare the performance of the intelligent controllers to that of the fluid flow approximation method. After that, a comparison between the neural network-based and the neuro-fuzzy-based bandwidth controllers.

The first set of experiments compares the bandwidth amount allocated to different sources with different network configuration. The first example examines the amount of bandwidth allocated for ten sources, where  $\lambda = 0.3$ , using the three techniques. The normalized effective bandwidth is plotted against several quality of services (i.e., the cell loss probability). The exact solution and the approximation are obtained from section 3. Figure 10.44 depicts the result of the neuro-fuzzy controller, while Figure 10.45 shows the



**Figure 10.44:** The bandwidth of ten sources, with  $\lambda = 0.3$ , using the neuro-fuzzy-based controller

result of the neural network-based controller compared to the fluid flow and fluid flow approximation models.

One can see that the performance of the proposed methods, in allocating the suitable amount of bandwidth, is better than the results obtained using the conventional approximation method. The precision of the proposed methods can be seen in cases of low cell loss probabilities where the fluid flow approximation method allocates a larger amount than the required bandwidth. The merit of these methods is that they lead to a better network resource utilization and reduce the amount of wasted bandwidth. In the fluid flow approximation method, the effective bandwidth is taken to be the sum of all individual sources since it is a more conservative estimate of the required bandwidth amount [24].

The second example illustrates the amount of bandwidth assigned for 35 sources, while varying the QoS (cell loss probability, CLP) criterion and setting the values of  $\lambda$  to 0.4. Figure 10.46 summarizes the results using the neuro-fuzzy-based bandwidth controller, while Figure 10.47 depicts the output of the neural network-based controller along with the results obtained from the fluid flow model and the approximation method. Again, the examples show that the intelligent controller's performance, in terms of the required amount of bandwidth, is better than that of the approximation method.

The next examples demonstrate the behavior, as a function of the QoS criterion (CLP), of the proposed methods and the fluid approximation method when the number of sources is 45 and  $\lambda$  is set to the value of 0.5. Figure 10.48 considers the performance of the neuro-fuzzy-based bandwidth allocation controller compared to the fluid approximation method

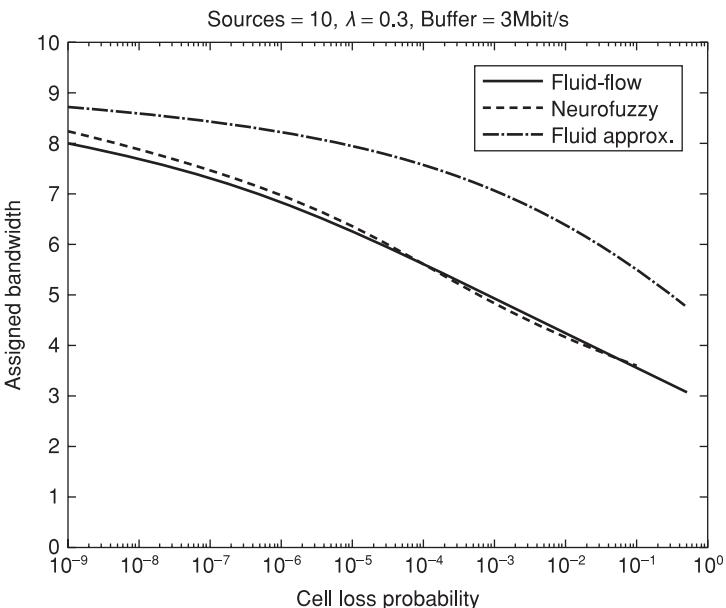


Figure 10.45: The bandwidth of ten sources, with  $\lambda = 0.3$  using the neural network-based controller

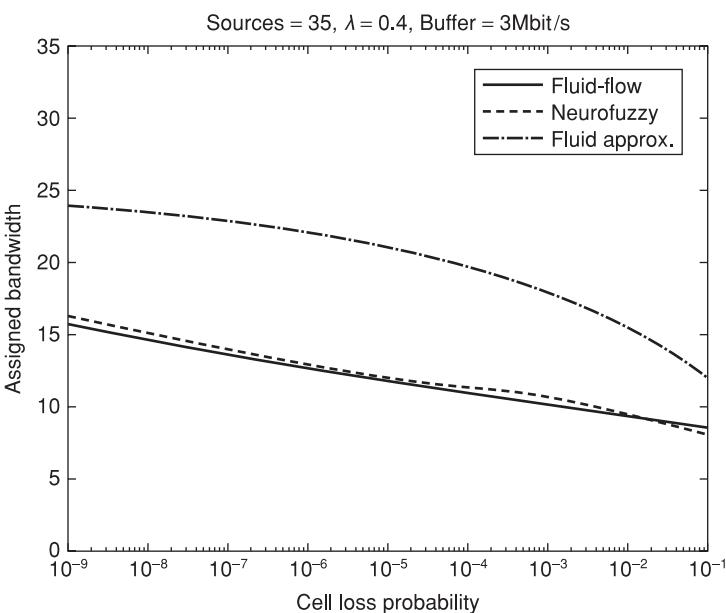
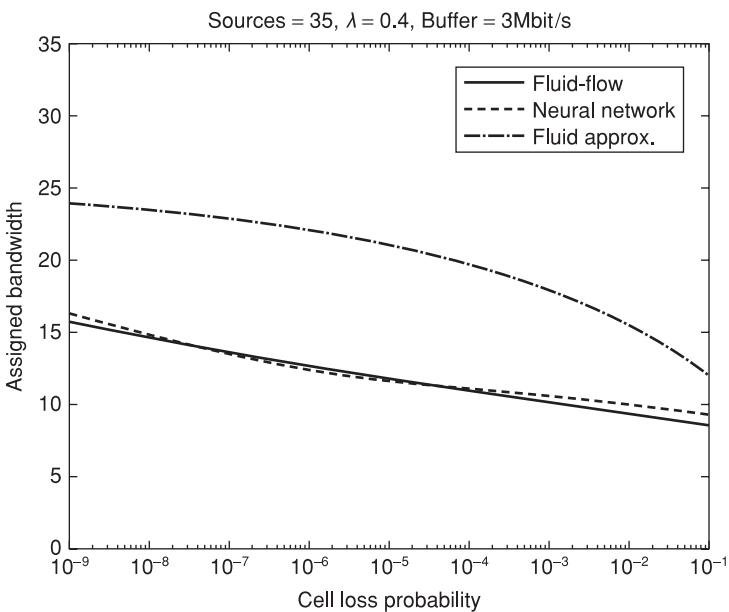
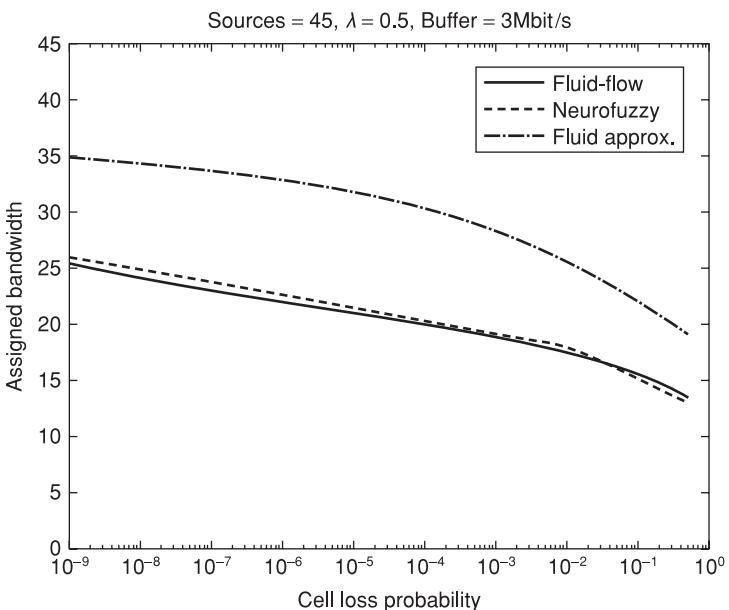


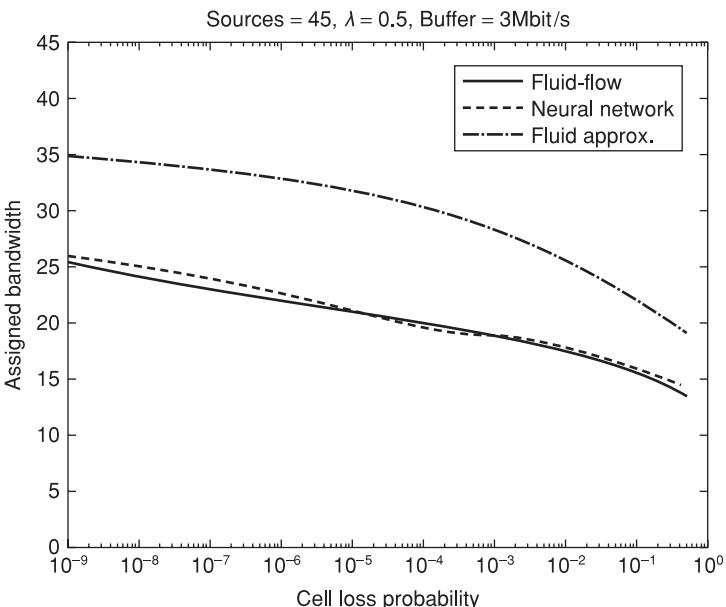
Figure 10.46: The bandwidth of 35 sources, with  $\lambda = 0.4$ , using the neuro-fuzzy-based controller



**Figure 10.47:** The bandwidth of 35 sources, with  $\lambda = 0.4$  using the neural network-based controller



**Figure 10.48:** The bandwidth of 45 sources, with  $\lambda = 0.5$  using the neuro-fuzzy-based controller



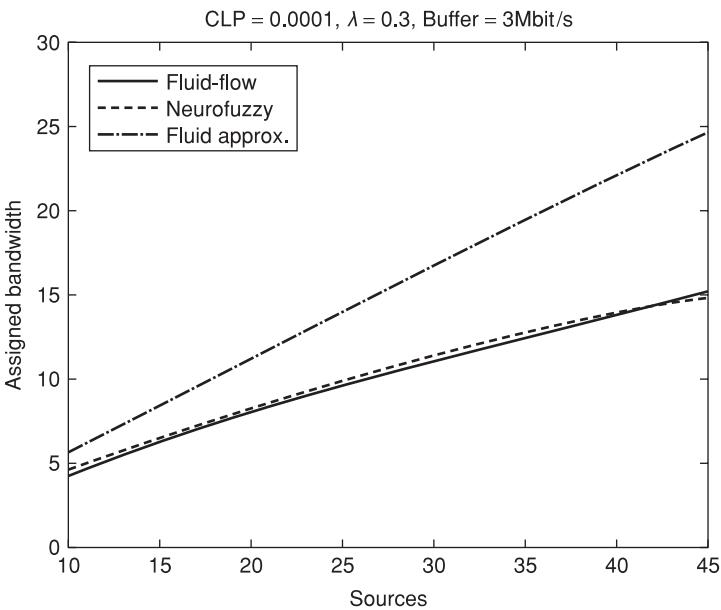
**Figure 10.49:** The bandwidth of 45 sources, with  $\lambda = 0.5$  using the neural network-based controller

while Figure 10.49 considers the performance of the neural network-based controller.

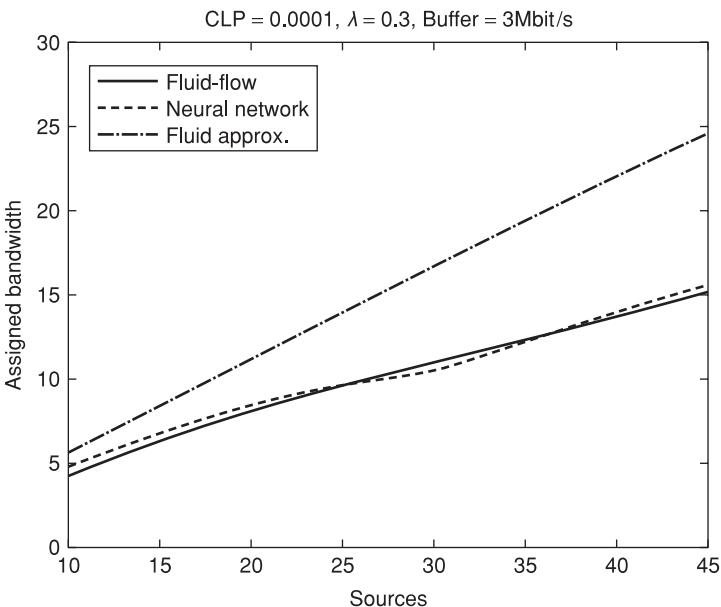
The investigation of the above experiments indicates that the bandwidth allocation schemes based on neural networks and neuro-fuzzy techniques have the required precision to assign the amount of bandwidth that is close to the exact solution. However, one can see that the bandwidth allocation scheme based on a neural network has an undesirable feature. For some cases, the neural network-based controller underestimates the required amount of bandwidth which may lead to the blocking of some sources while there is a sufficient amount of residual bandwidth.

The second set of experiments illustrates the performance of both the neuro-fuzzy and the neural network-based bandwidth allocation controllers for several sources. While the first set of experiments investigates the impact of changing the QoS criterion (CLP) with a given value of  $\lambda$ , the second set fixes the QoS criterion and varies the number of sources for a given value of  $\lambda$ . The first experiment in this set considers the amount of bandwidth for different sources when the CLP is set to the value of  $10^{-5}$  while  $\lambda$  takes the value of 0.3. Figures 10.50 and 10.51 illustrate the output of the neuro-fuzzy-based and neural network-based bandwidth controllers compared to the outputs of the fluid flow and the approximation models.

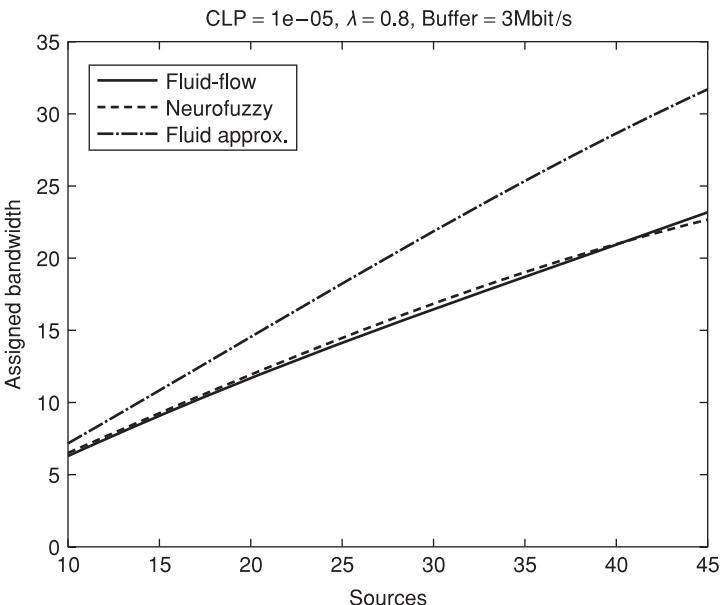
It is clear from the experiment results that the intelligent controllers perform better than the approximation method, especially when the number of sources becomes relatively large (i.e., greater than 20). This is due to the fact that the fluid flow approximation method computes the total bandwidth



**Figure 10.50:** The bandwidth for  $\lambda = 0.3$ , and overflow probability =  $10^{-5}$  using the neuro-fuzzy-based controller



**Figure 10.51:** The bandwidth for  $\lambda = 0.3$ , and overflow probability =  $10^{-5}$  using the neural network-based controller



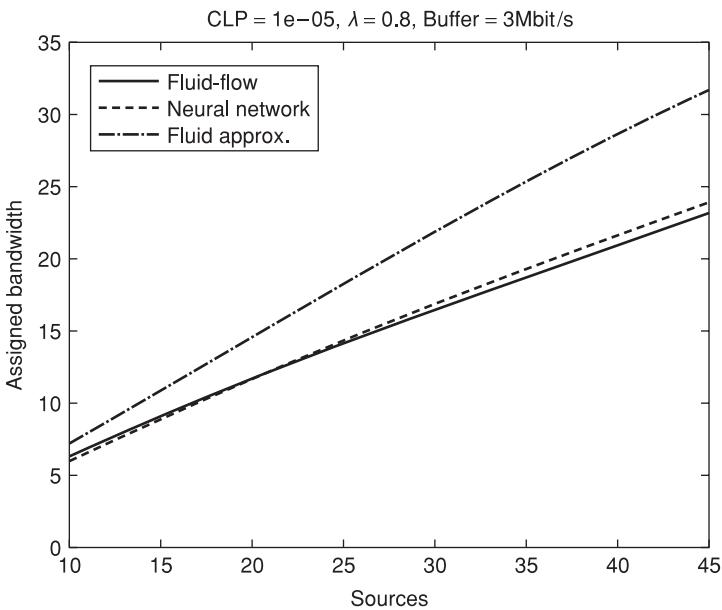
**Figure 10.52:** The bandwidth for  $\lambda = 0.8$ , and overflow probability =  $10^{-6}$  using the neurofuzzy-based controller

by multiplying the bandwidth amount of a single source by the number of sources while neglecting the multiplexing gain. Another example here is depicted in Figures 10.52 and 10.53 for both neuro-fuzzy and neural network-based controllers, respectively. This example illustrates the performance of the intelligent controllers compared to the conventional one when the CLP is set to the value of  $10^{-6}$  while  $\lambda$  take the value of 0.8.

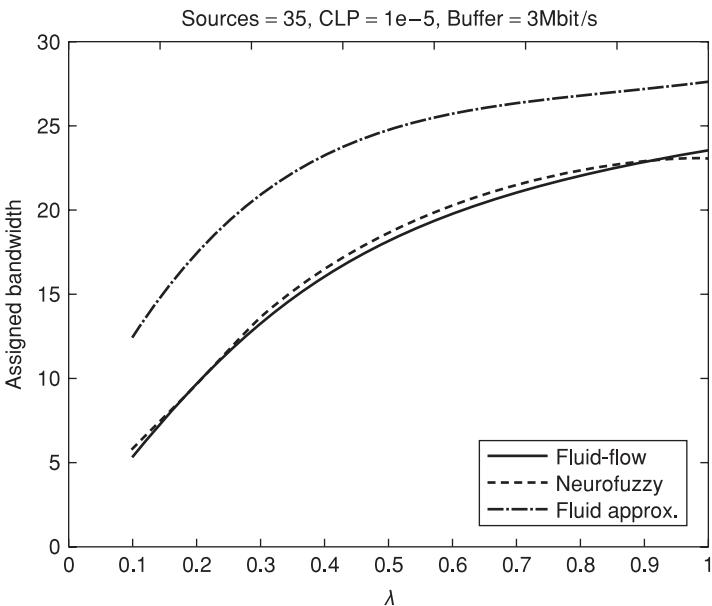
In the third set, the objective is to investigate the impact of changing  $\lambda$ , the rate of transition out of the OFF state for the sources, on the performance of the intelligent controllers. Here, we fix the the number of sources (i.e., 35) and vary the values of  $\lambda$  under a certain QoS criterion value of  $10^{-6}$ . Figure 10.54 illustrates the performance of the neuro-fuzzy-based bandwidth allocation controller compared to that of the conventional ones. Similarly, Figure 10.55 considers the performance of the neural network-based controller. Again, one can see that the soft computing-based controller outperforms the conventional one. Furthermore, the problem of under-estimated bandwidth by the neural network-based controller can be noticed in Figure 10.55.

## 6 Conclusion

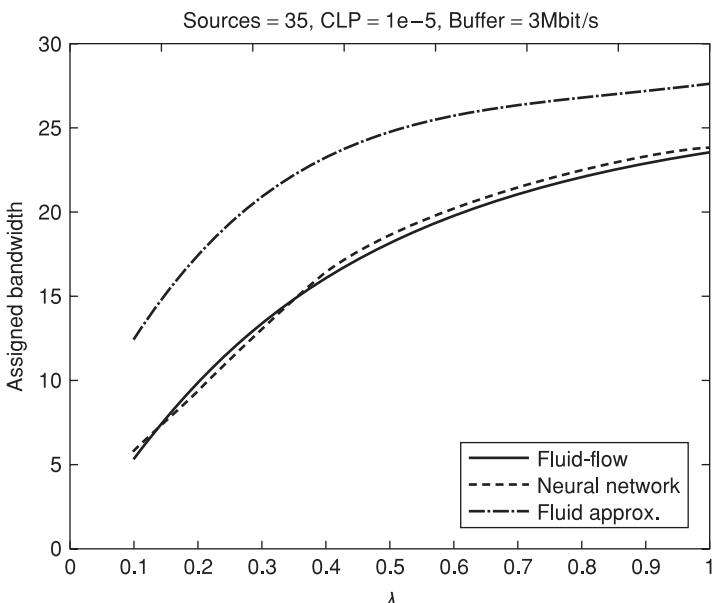
A set of dynamic bandwidth allocation controllers based on neural networks and neuro-fuzzy systems are presented here. Both controllers accurately assign the bandwidth required by set of sources of homogeneous classes to guarantee the QoS criterion. The controllers can deal effectively with a wide variety of traffic classes of different quality of service. When a new connection



**Figure 10.53:** The bandwidth for  $\lambda = 0.8$ , and overflow probability =  $10^{-6}$  using the neural network-based controller



**Figure 10.54:** The bandwidth for 35 source, and overflow probability =  $10^{-6}$  using the neuro-fuzzy-based controller



**Figure 10.55:** The bandwidth for 35 sources, and overflow probability =  $10^{-6}$  using the neural network-based controller

joins the network within a specific class, the intelligent controller adapts itself and efficiently finds the optimal bandwidth for the class. Preliminary results show accurate performance of the neuro-fuzzy and neural network models in estimating the required bandwidth. This leads to higher utilization of network resources. Generalization of the models to handle the variety of peak cell rates could be an aspect of further study.

## References

1. Jeffrey, M. (1994) “Asynchronous transfer mode: the ultimate broadband solution,” *Electronics and Communications Engineering Journal*, vol. 30, no. 6, pp. 143–151.
2. Sato, Y., and Sato, K. (1991) “Virtual path and link capacity design for ATM networks,” *IEEE Journal on Selected Areas in Communication*, vol. 9, no. 1, pp. 968–998.
3. Cheng, R., Chang, C., and Lin, L. (1999) “A QoS-provisioning neural fuzzy connection admission controller for multimedia high-speed networks,” *IEEE/ACM Transactions On Networking*, vol. 7, no. 1, pp. 111–121.
4. Jensen, D. (1994) “B-ISDN network management by fuzzy logic controller,” in *IEEE GLOBECOM 94*, San Francisco, CA, pp. 799–804.
5. Schwartz, M. (1996) *Broadband Integrated Networks*, Prentice Hall, Englewood Cliffs, NJ.
6. Guerin, R., Ahmadi, H., and Nagshienh, M. (1991) “Equivalent capacity and its applications to bandwidth allocations in high speed networks,” *IEEE Journal on Selected Areas in Communication*, vol. 9, no. 7, pp. 968–998.

7. Brown, M., and Harris, C. (1994) *Neurofuzzy Adaptive Modelling and Control*, Prentice Hall, Englewood Cliffs.
8. Daigle, J.N., and Langford, J.D. (1986) "Models for analysis of packet-voice communication systems," *IEEE Journal on Selected Areas in Communication*, vol. SAC-4, no. 6, pp. 847–855.
9. Tsern-Huei Lee, Kuen-Chu Lai, and Shii-Tyng Duann (1996) "Design of a real-time call admission controller for ATM networks," *IEEE/ACM Transactions On Networking*, vol. 45, no. 10, pp. 758–765.
10. Kelly, F.P. (1991) "Effective bandwidths at multi-class queues," *Queueing Systems*, vol. 9, pp. 5–16, 1991.
11. Kesidis, G. (1994) "Modeling to obtain the effective bandwidth of a traffic source in an ATM network," *MASCOTS'94, Modeling Analysis and Simulation of Computer and Telecommunication Systems*, Los Alamitos, CA, pp. 318–322.
12. Kesidis, G., Walrand, J., and Chang, C. (1993) "Effective bandwidths for multi-class Markov fluids and other ATM sources," *IEEE/ACM Transactions On Networking*, vol. 1, pp. 424–428.
13. Elwalid, A., and Mitra, D. (1993) "Effective bandwidth of general Markovian traffic sources and admission control of high-speed networks," *IEEE/ACM Trans. On Networking*, vol. 1, no. 3, pp. 329–343.
14. Chang, C., and Thomas, J. (1995) "Effective bandwidth in high-speed digital networks," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 6, pp. 1091–1100.
15. Anick, D., Mitra, D., and Sondhi, M. (1982) "Stochastic theory of a data-handling system with multiple sources," *Bell Sys. Tech. J.*, vol. 61, no. 8, pp. 1871–1894.
16. El-Sayed, K., and Perros, H. (1996) "On the effective bandwidth of arbitrary ON/OFF sources," *Proceeding of the Sixth IFIP WG6.3 Conference on Performance of Computer Networks*, London, pp. 257–271.
17. Choudhury, G., Lucantoni, D., and Whitt, W. (1994) "On the effectiveness of effective bandwidths for admission control in ATM networks," *Proceedings of 14th International Teletraffic Congress ITC 14*, Amsterdam, pp. 411–420.
18. Jang, J.S.R., Sun, C.T., and Mizutani, E. (1997) *Neuro-Fuzzy and Soft Computing*, Prentice Hall, Englewood Cliffs.
19. Lin, C.T., and Lee, C.S. (1996) *Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems*, Prentice Hall, Englewood Cliffs.
20. Jang, J.S.R. (1993) "ANFIS: adaptive-network-based fuzzy inference system," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 23, no. 3, pp. 665–685.
21. Douligeris, C., and Develekos, G. (1997) "Neuro-fuzzy control in ATM networks," *IEEE Communications Magazine*, vol. 35, no. 5, pp. 154–162.
22. Ghosh, S., and Razouqi, Q. (1998) "A survey of recent advances in fuzzy logic in telecommunications networks and new challenges," *IEEE Transactions on Fuzzy Systems*, vol. 6, no. 3, pp. 443–447.
23. Hiramatsu, A. (1991) "Integration of ATM call admission control and link capacity control by distributed neural networks," *IEEE Journal on Selected Areas in Communication*, vol. 9, no. 7, pp. 443–447.
24. Zhong Fan and Mars, P. (1996) "Application of artificial neural networks to effective bandwidth estimation in ATM networks," *Proceedings of International Conference on Neural Networks (ICNN'96)*, New York, NY, pp. 1951–1956.



# Index

- action frames 18  
adaline (adaptive linear neuron) model 242–4  
adaptive control 311–13  
adaptive time delay neural networks 328  
adaptive vector quantization 422  
Adeli-Hung algorithm 342–3  
air conditioning 61  
airbag release in cars 401  
alleles 367, 374  
 $\alpha$ -cut of a fuzzy set 90  
Amjadi, N. 285  
AND operator 72  
ANFIS (adaptive-network-based fuzzy inference system) 347–54, 359  
Anick, D. 536  
antilock braking systems 62  
approximation 45–50, 105  
artificial intelligence (AI) 5–6, 17, 58, 138  
artificial neural networks (ANN) *see also* neural networks  
associativity 122–3  
asymptotic stability 181  
ATM (asynchronous transfer mode) networks 287, 533–9  
Auda, G. 289  
backpropagation learning (BPL) algorithm 227, 250–1, 254, 283, 358–9, 485–7  
backpropagation through time 304–6  
backward chaining 16–17  
Bayesian analysis 44  
belief, levels of 48–50  
Bezdek, J.C. 356  
BIBO (bounded-input-bounded-output) stability 181  
Biesterfeld, J. 287  
“black box” structure 338  
blackboard systems 20–2  
Boolean algebra 97–8, 128  
brain function 223  
camcorders 62  
Canon (company) 62  
capacitated concentrator location 288  
Card, H.C. 343  
Cartesian product of fuzzy sets 100–1  
cascade correlation alarm correlator 288  
CDMA (code division multiple access) networks 522–3  
centroid method of defuzzification 151–3, 449  
chaining, forward and backward 16–17  
chaos prediction 325–8  
characteristic functions 66  
Chen, C. 284  
Choi, J.Y. and C.-H. 326  
chromosomes 367, 373  
clustering, fuzzy 356  
commutativity 121–2  
competitive learning 43  
complement of a set 29–30, 69–70  
completeness of a rule base 181  
composition in fuzzy relations 116–26  
compositional rule of inference (CRI) 105, 117–20, 128, 141, 146–7, 153, 417–18, 421, 446–8  
conflict resolution 17  
conjunction of propositions 31, 72  
connection admission control (CAC) 286, 533–4  
connectionist modeling 233–4, 249, 289, 308; *see also* adaline model  
consistency of a rule base 182–5  
content-addressable memory 274

- continuity of a rule base 181–2  
control actions of operators 444–5  
control algorithms 138–41  
controller tuning 413–27  
cooperative modular neural networks 289  
cooperative neuro-fuzzy systems 340–2  
coupled rule bases 187–8  
crisp logic 28–37  
crossover 377–8, 382–3, 389  
Cui, X. 285–6  
Cybenko, G. 250  
cylindrical extension 105, 109–15
- databases 13  
deception (in genetic algorithms) 392  
decision-making  
  through coupled and uncoupled rule bases 187–90  
  fuzzy 126–8  
decision tables 176–80  
decoupling of a rule base 186–91  
defuzzification 127–8, 151–3, 202  
delta functions 146  
DeMorgan’s Laws 123  
Dienes–Rescher implication 84–5, 89  
differential competitive learning 422  
direct-digital-control loops 141  
disjunction of propositions 31, 70–1  
distributivity 123  
“don’t care” symbols 375–6  
Drucker, H. 289  
dynamic systems 191–2, 299–301  
  for chaos time-series prediction 323–8  
  for identification and control 307–13
- Ehsan, M. 285  
eigen fuzzy sets 200–2  
elevator control 62  
Elman network 302  
El-Sayed, M. 285  
El-Sharkawi, M. 284  
epistasis 392–3  
equivalence conditions for control  
  inferences 190–1  
European Commission 522  
evolution cycle 368  
evolutionary algorithms 368–9  
evolutionary computing 43, 365–72  
evolutionary programming 282, 369–70  
evolutionary strategies 370, 395–9  
expert systems 12, 15, 22–6  
  fuzzy logic in 64–5
- expertise, human 8–9, 444  
extension principle 101–5, 125–6
- facilities layout planning (case study) 510–21  
Farag, W. 284  
Federal Communications Commission 522  
feedback/filter modules 168–9  
feedback linearizing control (FBLC) 497–509  
feedforward networks 250, 263–4, 299, 325–6  
feedforward topology 227–8  
finite state machines (FSMs) 370  
fish processing machinery 427–43, 463–7  
Fisher (company) 62  
fitness functions 374–5, 484  
five-layer neuro-fuzzy systems 347–9  
Fogel, D. 369  
forward chaining 16  
forward production systems 16  
four-layer neuro-fuzzy systems 350  
frame-based systems 17–20  
free launch theorem 396  
Freund, Y. 289  
Fujitec (company) 62  
functional/control modules 169  
fuzzification 137, 143, 146, 153–6, 202
- fuzziness  
  degrees of 93  
  effect on processing of 169–71  
  effect of signal combination on 169–71  
  measures of 93–7  
fuzzy associative memory (FAM) 422  
fuzzy control 137, 140–5, 202, 445  
  architectures 162–5  
  properties of 180–91  
fuzzy control surface 156–7  
fuzzy decision-making 126–8  
fuzzy logic 32, 38–41, 45–50  
  applications of 61–3  
  basic laws of 72–5  
  evolution of 60–3  
  in expert systems 64–5  
  generalized operations in 76–7  
  integration with genetic algorithms 390–1  
  Japanese dominance in 63–4  
  limitations of 337–9  
  operations in 68–75

- fuzzy logic control (FLC) 145–7, 162  
 fuzzy relations 97–105  
   analytical representation of 99–100  
   tables of 175–6  
 fuzzy resolution 91–3  
 fuzzy sets 65–8, 337–8  
   Cartesian product of 100–1  
   definitions and concepts of 89–91  
 fuzzy tuners 172–6
- GANN (genetic algorithms with neural networks) 390  
 Gaussian function method of fuzzification 155–6  
 genes 367, 373–4  
 genetic algorithms (GAs) 43–5, 371–2, 401, 482–4  
   drawbacks of 391–3  
   fundamental theorem of 375–6  
   integration with fuzzy logic 390–1  
   integration with neural networks 388–90  
   mode of operation 378–80  
   operators 376–8  
   and optimization 372–5  
   search process 381–3  
   steps in implementation of 381  
 genetic programming 370–1  
 genotypes 373–5  
 GESA (guided evolutionary programming with simulating annealing) 282  
 Gram–Schmidt process 326  
 Greenwood, I. 400  
 grid-partitioning, fuzzy 355–6  
 GSM mobile phone network 288  
 Guerin, R. 538, 544
- handheld computers 62  
 Hatanaka, T. 400  
 Hebbian learning 43, 237, 276  
 height of a fuzzy set 89  
 hierarchical structures and systems 137, 164–9, 202, 409–11  
 Hitachi (company) 62  
 Hoang, D. 288  
 Holland, J. 371–2, 375  
 Hopfield networks 42, 274–81, 287–8  
   applications of 280–1  
 Hsu, Y. 284  
 Huang, S. 289  
 hybrid learning algorithms 359  
 hybrid neuro-fuzzy systems 345–53  
 hypothetical syllogism rule of inference (HSRI) 35–6
- if-then statements 82, 337  
 image processing 400  
 implications 82–9  
 inclusion property 123–5  
 incremental learning 393–5  
 Industrial Automation Laboratory, University of British Columbia 60–1, 429  
 inference, rules of 35–7; *see also*  
   compositional rule of inference  
 inference engines 13–15, 23  
 Information Engineering Division, University of Cambridge 60  
 inf-S composition 121  
 integrated circuits 282  
 intelligence 6–7  
   dynamics of 8–9  
   human 50–1  
 intelligent controllers 139–41, 314, 408–9  
 intelligent machines 9–12, 408–11  
 intelligent processes, structure of 10–11  
 intelligent products, development of 63–4  
 intelligent systems 3–12  
 intersection of sets 30, 72  
 inverse controllers 321
- Jacobs, R.A. 289  
 Jang, J.S.-R. 359  
 Japanese dominance in fuzzy logic 63–4  
 “join” in fuzzy relations 105, 115  
 Jordan’s sequential network 302
- Kamel, M. 289  
 Kim, B. 283  
 knowledge acquisition facilities 23–4  
 “knowledge base” concept 9, 13, 23–4  
 knowledge-based systems (KBSSs) 12–26, 58, 139–40, 308, 446  
   architecture of 14–15  
   basic structure of 13  
 knowledge engineering 25  
 knowledge systems 9, 408–11  
 Kohonen self-organizing network (KSON) 268–74, 285  
   applications of 273–4  
 Kraft, T. 284
- Larsen implication 84–6, 89  
 learning, supervised and unsupervised 230–2

- learning algorithms 224–7, 230–3, 249–50, 358  
hybrid 359
- learning rate (in PBIL algorithm) 395
- Levenberg–Marquardt algorithm 498, 542
- Linkens, D. 359
- LMS (least mean square) algorithm 242–3
- local minima in complex sets 391–2
- logic 30–5
- Lorenz, E.N. 323
- Louchet, J. 400
- Lukasiewicz implication 84–5, 89
- Lyapunov methods 193–4
- McCulloch–Pitts models 233–4
- machine intelligence 5–6, 26, 45
- machine tuning 453, 461–3
- Mackey–Glass technique 324, 327
- madaline model 244–5, 249
- Mamdani model 84–6, 89, 127–8, 487  
extensions of 162
- Matsushita (company) 62
- May, G. 283
- “mean of maxima” method of defuzzification 150–3
- membership functions 57, 66, 128, 141, 341–2
- Meng, R. 286
- meta-rules 445
- micro-electromechanical systems 408
- Minsky, M.L. 6, 242
- Mitra, D. 536
- Mitsubishi (company) 61
- model-referenced adaptive control 311–13
- modular development of software 455–6
- modular neural networks 288–9
- modus ponens* and *modus tollens* rules of inference 35–6
- momentum 260–2
- motor controllers (case study) 476–95
- multilayer perceptron (MLP) model 250–63  
applications of 262–3
- mutation 378, 382–3
- mutation interference 392
- Narendra, K. 325
- negation of a proposition 30–1, 69–70
- neural networks 41–5, 223–5, 249–50, 289–90  
activation functions 228–30
- case studies of 484–7, 497–501
- for chaos prediction 325–8
- in communications 286–8
- control approaches based on 313–23
- in decision fusion 288–9
- features of 226–33
- fuzzy reasoning driven by 344–5
- for identification 315–17
- industrial and commercial applications of 281–9
- integration with genetic algorithms 388–90
- learning algorithms 230–3
- in pattern recognition 288–9
- for power systems 284–5
- research on 358
- in robotics 285–6
- topologies of 227
- traffic enforcement mechanism 287
- weaknesses of 337–9
- neuro-adaptive control 322–3
- neuro-fuzzy systems 337–8  
architecture of 339–53  
classes of 338–9  
construction of 355–9  
cooperative 340–2  
hybrid 345–53  
parameter learning phase 357–9
- Nissan (company) 61–2
- NOT operator 69–70
- nuclear power plants 388–9
- numerical learning 224–6
- objective functions 374–5
- object-oriented programming 22
- OGY method of system control 325
- OR operator 70–1
- Ostertag, M. 401
- Panasonic (company) 61–2
- Papert, S. 242
- parallel model for neural identification 316–17
- parameter estimation 400–1
- Park, Y. 287
- Parthasarathy, K. 325
- partitioning, fuzzy 355–6
- pattern recognition 288–9
- Pedrycz, W. 343
- perceptron model 234–7, 242–3, 249

- performance testing of machines 457–67  
 Picton, P.D. 286  
 Pitts, W. 233; *see also* McCulloch–Pitts models  
 Platt, J. 326  
 population-based incremental learning 393–5  
 possibility functions 66  
 power systems 284–5, 388–9, 497–509  
 predicate calculus 37  
 premature convergence in complex sets 392  
 probabilistic reasoning 44–5  
 processing, effect of fuzziness on 169–71  
 production systems 15–17  
 projection, fuzzy 105–8  
 proportional-integral derivative (PID) control 138, 418, 422–7, 476–8  
 propositional calculus 37  
 prototypes 412  
 quality assessment for products 452–3, 463–7  
 RBF (radial basis function) networks 263–7, 326, 522–2  
     applications of 267  
 real time backpropagation learning 305–6  
 reasoning strategies 16–17  
 Rechenberg, I. 370, 395–6  
 recurrent networks 299–301  
     applications of 306–7  
     architecture of 301–4  
     dynamics of 301  
     predictors based on 327  
 recurrent topology 227–8  
 reinforcement learning 232–3  
 representation theorem 90, 105  
 resolution relations 419–21  
 robotics 285–6  
 robustness of control systems 191  
 Rosenblatt, F. 234–5, 249  
 rule base concept 415–16, 444–6, 451–2  
 rule base decoupling 186–91  
 rule dissociation 418–19  
 rule interaction 185–6  
 rule searching 15  
 rule validity 185  
 Rumelhart, D.E. 358  
 Sanyo (company) 62  
 scatter partitioning (of fuzzy rules) 357  
 schema theorem 375–6  
 Schwefel, H. 370  
 self-tuning schemes 312–13, 495  
 semantic networks 27–8  
 semiconductor manufacturing 282–4  
 sensory perception 9, 408  
 separability of fuzzy restrictions 190  
 servo tuning 449–52, 457–60  
 set equality 82  
 set inclusion 80–2  
 set theory 29–33  
 Shi, X. 286  
 Shibata, T. 286  
 Shin, K.G. 285–6  
 signal combination 169–71  
 simple recurrent networks 302  
 singleton method of fuzzification 154–5  
 situational frames 18  
 soft computing xvii–xviii, 38–51  
 Sondhi, M. 536  
 Sony (company) 62  
 Srinivasan, D. 285  
 stability of control systems 191–200  
 statistical quality control 282  
 Subaru (company) 61  
 subsets 30, 80  
 subway trains 62–3  
 Sugeno model 127–8, 162  
 supervised control 320  
 supervised learning 230–3, 276, 358  
 support sets 89–90  
 sup-product composition 116–17  
 sup-T composition 121  
 syllogism 36  
 symbolic learning 224  
 symbolic representation 66–8  
 synchronous updating 277  
 system identification 310–11  
 Takagi–Hayashi method of reasoning 344–5  
 Takagi–Sugeno fuzzy inference systems (TSFIS) 480–2, 487–8, 495  
 Takagi–Sugeno–Kang model *see* Sugeno model  
 task scheduling 400  
 teacher forcing 306  
 technology transfer 412–13  
 telephone networks 287–8

- television sets 62  
three-layer neuro-fuzzy approximator 350–1  
threshold methods of defuzzification 152  
time-delay neural networks 314–15, 327–8  
Toshiba (company) 60, 62  
training algorithms 304–6  
triangular function method of fuzzification 154–5  
triangular norms 77–80  
TSK model *see* Sugeno model  
tuning of controllers 413–27  
tuning protocols 172–3
- Uhrig, R.E. 388  
uncertainty, concept of 45–6  
union of sets 30, 70–1  
unsupervised learning 231–2, 358  
user interfaces 14, 456–7
- vacuum cleaners 61  
Venn diagrams 28–9, 65  
voting schemes 289
- Waldemark, J. 288  
washing machines 62  
Watanabe, K. 401  
Weersooriya, S. 284  
Werbos, P. 231, 249–50, 320  
Whitely, D. 389  
Widrow–Hoff learning rule 242  
Wienholt, W. 379  
wildcard symbol 375–6
- Yeh, I.C. 511  
Yousef, S. 287
- Zadeh, Lotfi xvii, 57, 60, 67, 84–5, 89, 101, 128, 190, 511  
Zhang, Y. 287  
Ziegler–Nichols methodology 413, 476