DISTRIBUTED MODELING

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

MASTER OF SCIENCE

BY

Patrick C. Lam

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

ISBN 0-315-81710-0

Name_____

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

ELECTRONICS AND ELECTRICAL

SUBJECT TERM

0544

SUBJECT CODE

UMI

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

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

EDUCATION

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

LANGUAGE, LITERATURE AND LINGUISTICS

Language
- General 0679
- Ancient 0289
- Linguistics 0290
- Modern 0291

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

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy 0422
Religion
- General 0318
- Biblical Studies 0321
- Clergy 0319
- History of 0320
- Philosophy of 0322

Theology 0469

SOCIAL SCIENCES

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

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

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

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

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

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

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

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

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

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

Biophysics
- General 0786
- Medical 0760

EARTH SCIENCES

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

HEALTH AND ENVIRONMENTAL SCIENCES

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

Home Economics 0386

PHYSICAL SCIENCES

Pure Sciences

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

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

Statistics 0463

Applied Sciences

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

Geotechnology 0428
Operations Research 0796
Plastics Technology 0795
Textile Technology 0994

PSYCHOLOGY

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

Nom _____

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

_____ ☐☐☐ U·M·I

SUJET — CODE DE SUJET

Catégories par sujets

HUMANITÉS ET SCIENCES SOCIALES

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

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

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

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

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

Théologie 0469

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

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

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

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

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

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

Transports 0709
Travail social 0452

SCIENCES ET INGÉNIERIE

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

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

Biophysique
- Généralités 0786
- Medicale 0760

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

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

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

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

Statistiques 0463

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

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

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

DISTRIBUTED MODELING

BY

PATRICK C. LAM

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

MASTER OF SCIENCE

I hereby declare that I am the sole author of this thesis. I authorize the University of Manitoba to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Manitoba to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Patrick Lam

1993

The University of Manitoba requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

To Catherine, Vivian and Rachel

for their loves, supports and encouragement

in all these years.

Abstract

This thesis deals with distributed modeling, and its basic building block - the Logic Processor (LP). The aim of distributed modeling is to depart from a single analytical and fully centralized model and build a family of loosely coupled local models based on the LP to describe the performance of a considered system. Aspects of fuzzy sets theory and neural network are combined to develop the LP. A new set of logic-oriented neurons based exclusively on triangular norms proposed in the fuzzy sets theory is used to build the LP. The LP is constructed as a three layered heterogeneous fuzzy neural network, with each layer realizing different functions. The architecture of the LP is organized in order to carry out an approximation task and formulates the logical relationship between the system inputs and its state variables, which can be used to reason about the system later on. Details of the construction of the LP, from the performance indices, to the architecture of the LP are included. A discussion on the learning algorithm for the LP is presented. Several software simulations of the LP and distributed modeling are also provided including a classic Box-Jenkins data set.

Acknowledgments

I would like to express my sincere thanks to my advisor, Dr. Witold Pedrycz for his kind assistance, guidance and support for this thesis. Never has a graduate student had a finer advisor. Thanks also to my colleagues who patiently read and commented on drafts of chapters. In particular I would like to thank Rachel Choo, Sunny Lam and Scarlet Shaw.

This work was done through equipment loans from the Canadian Microelectronics Corporation.

Contents

List of Tables

List of Figures

Chapter 1
Introduction

1.1 Problem

The manipulation of knowledge and the use of this knowledge to reason about the physical world are the keys to model any sophisticated intelligent agent. The goal of the naive physics [6] is to represent the commonsense knowledge people have about the physical world. Human beings appear to use a qualitative causal calculus with commonsense knowledge in reasoning about the behavior of their physical environment [3]. One of the main issues in qualitative reasoning is to find a formal representation of knowledge suitable to perform reasoning about the physical world.

The implementation of artificial intelligence can be categorized into two main streams: manipulation of rules and symbolic symbols; and utilization of the learning process of a human being from modeling the brain [5]. The former method is based on symbolic symbols to encode knowledge and manipulate rules of a problem domain. It works fine for applications in which the environment is known and predictable. The later implementation is based on numerical manipulation. Knowledge is stored through some numerical values in the connections of the neurons in a neural network. However, this knowledge is implicitly encoded in the network which makes it difficult to translate and

interpret. Without a suitable representation of the knowledge, it is hard to understand what has been encoded during the learning process.

The above two implementations share a common purpose, that is, to build a machine that has intelligence. In system modeling and identification process, we are usually provided with a vast amount of experimental data, and is required to describe a model that can govern the physical phenomenon. In many cases, the structure of the system is too complex, and it is either too difficult to identify the model, or it is too hard to build a model that can be easily understood. Without an understanding of the knowledge underlying the system, we can not do further analysis of the system. Therefore, we need to find a way which is general enough to capture the knowledge of a system, and translate that into a format that we can easily use to reason with.

1.2 Purpose

The main purpose of this thesis is to examine the knowledge representation capabilities in a fuzzy neural network, called the Logic Processor (LP) and one of its potential application - distributed modeling. The learning algorithm of the LP and its ability to translate the underlying knowledge captured in the LP are also studied.

LP is a major component used in distributed modeling. Roughly speaking, distributed modeling establishes relationships among the inputs and the state variables in a system by a family of loosely coupled local models. Each of these local models is associated with a LP to model one state variable in the system, its task is to describe the

logical interactions with other participated state variables and inputs in the considered system.

Fuzzy sets theory involved in this thesis refer to the work done by L. Zadeh [27]. Fuzzy logic is an extension of multivalued logic, and is a formal representation to handle concepts with a gradual membership character.

Neural network is inspired by the structure of neurons, and their interconnections within the human brain. It realizes a non-linear mapping between the input and output spaces. Neural network is a structurally massive parallelism network with the basic local processing elements. These basic local processing elements are called neurons, and are arranged into layers. Processing elements are connected by links, and each link is associated with a weight. Inputs to the neural network are propagated through the layers, and outputs are determined with the combination of the weights stored in the network described by some mathematical equations.

The formal structure of fuzzy sets can be viewed as an interesting tool in mapping the logic properties of some problems into the topologies of neural network. Combining the fuzzy sets theory and the neural network, we can construct a new class of neural network, called fuzzy neural network. The fuzzy neural network uses a set of logic-oriented model of neurons instead of the typical neurons found in the existing neural networks. The construction of these new logic-oriented neurons are based exclusively on triangular norms (t- and s-norms) proposed in the theory of fuzzy sets as a model of logic set connectives (*AND, OR*). The neurons in the fuzzy neural network are arranged in a logic architecture, and this enhances the representation power where the underlying logic structure of the problem can be captured in the network architecture.

1.3 Scope

This thesis covers the introductory theory of the Logic Processor (LP) used to build distributed models. Although a large number of diverse fields of study are incorporated in the research, only the relevant information will be covered.

The main body of this thesis includes the details of the construction of the LP, and the description of distributed modeling. This thesis is structured into five chapters. Chapter 2 gives an overview on the subjects related to this thesis, including the fuzzy sets theory, neural networks and fuzzy neural networks. The new logic-oriented models of neurons are also introduced in this chapter. Chapter 3 describes the logic processor. The architecture of the LP, and the performance indices are included. A detailed discussion on the learning algorithms for the LP is presented. Translation of weights stored in the LP to a logical knowledge representation is also illustrated. Two examples of software simulations of LP are included at the end of the chapter. In Chapter 4, we discuss the distributed modeling. Concepts on the construction of distributed modeling is introduced here. An example on developing a distributed model for a gas furnace system (the classic Box-Jenkins data set) is shown. The final conclusions and recommendations are given in Chapter 5. Appendix A shows the derivations of the learning algorithms for the LP. Appendix B gives the program listings used for the software simulations. The data of the gas furnace system used for the illustration of distributed modeling is listed in Appendix C and the weights in the LPs for the gas furnace model are shown in Appendix D.

Chapter 2
Overview

2.1 Overview of Fuzzy Sets Theory

Fuzzy logic theory deals with the mathematical theory of inference with vague or ambiguous concepts. Actually, the name of 'Fuzzy Logic' is a little misleading since there is nothing 'fuzzy' about the logic. It is indeed as precise as the traditional methods of inference, but applied to a concept that is ambiguous, unlike the two-valued logic.

In the classical two-valued logic and set theory, all sets are assumed to have sharply-defined boundaries. So any object expressed in a given universe of discourse (space) either belongs to a set, or is completely excluded in that set. Qualifications like 'alive-dead' and 'odd-even numbers' are examples of concepts with sharply-defined boundaries, which are suitably represented by the two-valued logic. However, most of the sets we are faced in everyday activity do not possess sharply-defined boundaries. The concepts like 'heavy', 'tall', 'pretty' and 'expensive' are good examples of objects with ill-defined boundaries. The classical two-valued logic does not suitably represent these types of concepts.

On the other hand, fuzzy sets theory is designed to deal with sets that do not have sharply-defined boundaries. Each object expressed in a given space is characterized by a

degree which it belongs to a certain set. This degree is well-known as a membership value. The more an object shares a given property characteristic for the concept, the closer the membership value is to 1. Conversely, the less the object shares the property of the concept, the closer the membership value is to 0.

2.2 Background of Fuzzy Logic

In the early 1930s, Polish logician Jan Lukasiewicz developed a three-valued logical system [21]. He then extended the range of the three-valued logic from {0, 1/2, 1} to all rational numbers in [0, 1]. However, the multivalued logic was derived too early to attract any technological interest at that time. In 1965, scientist Lotfi Zadeh at the University of California at Berkeley published the paper *Fuzzy Sets* [27] that formally developed the idea of fuzzy sets theory, and paved the road for applying this theory to solve real-world tasks. Today, the Japanese had applied the fuzzy logic technique to a variety of commercial and industrial products, from camcorders to subway control processes [23].

2.3 Mathematical Overview of Fuzzy Sets

The aim of this section is to provide a mathematical background for later chapters and therefore, only the pertinent material of the fuzzy sets theory, and fuzzy relational calculus will be included. The interested reader may refer to [4][26] for a detailed discussion of the fuzzy sets theory.

Notations and Terminology

Throughout the thesis, the following notations will be used. Let $\mathbf{X}$ represent the universe of discourse (space). μ_A will denote the membership function of fuzzy set A, especially $\mu_A(x)$[1] will stand for the membership value of object x in fuzzy set A.

2.3.1 Fuzzy Sets Theory

A fuzzy set A is a set of ordered pairs,

$$A = \{(x,\ \mu_A(x))\}, \qquad x \in \mathbf{X} \tag{2.1}$$

where $\mu_A(x)$ is the degree of membership of x in set A, and x is an object in $\mathbf{X}$.

The fuzzy set A can also be viewed as the union of the ordered pairs, which is represented in the form

$$A = \mu_A(x_1)/x_1 + \mu_A(x_2)/x_2 + \cdots + \mu_A(x_n)/x_n \tag{2.2}$$

Example 1

Let us discuss a discrete (finite) universe of discourse $\mathbf{X}$ as specified.

$$\mathbf{X} = 0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 \tag{2.3}$$

---

1 or $A(x)$ as shown in some other references. However, we will adopt this symbol $\mu_A(x)$ as the membership value of object x in set A, and use it throughout the document.

The fuzzy set *low* is the collection of the ordered pairs as below:

$$low = \{(0, 1), (1, 1), (2, 0.8), (3, 0.6), (4, 0.4),(5, 0.2)\} \tag{2.4}$$

or $$low = 1/0 + 1/1 + 0.8/2 + 0.6/3 + 0.4/4 + 0.2/5 \tag{2.5}$$

with the understanding that in (2.4) and (2.5), only the elements with non-zero membership values are listed.

Some basic operators on fuzzy sets are given below:

Union: The union of two fuzzy sets A and B with respective membership functions μ_A, μ_B is a fuzzy set C, written as $C = A \cup B$, whose membership values are related to those of A and B by

$$\mu_C(x) = \mu_{(A \cup B)}(x) = Max[\mu_A(x), \mu_B(x)], \qquad x \in \mathbb{X} \tag{2.6}$$

Intersection: The intersection of two fuzzy sets A and B with respective membership functions μ_A, μ_B is a fuzzy set C, written as $C = A \cap B$, whose membership values are related to those of A and B by

$$\mu_C(x) = \mu_{(A \cap B)}(x) = Min[\mu_A(x), \mu_B(x)], \qquad x \in \mathbb{X} \tag{2.7}$$

Complement: The complement of a fuzzy set A is denoted by $\overline{A}$, and its membership values are defined as:

$$\mu_{\overline{A}}(x) = 1 - \mu_A(x) \qquad x \in \mathbb{X} \tag{2.8}$$

2.3.2 Triangular Norms

The two triangular norms, t-norm and s-norm denoted by t and s respectively, are defined as follows:

By t-norm, we mean a two-place function which satisfies the conditions

$$t:[0,1]\times[0,1] \rightarrow [0,1]$$

(i) commutativity:
$a\ t\ b = b\ t\ a;$ (2.9)

(ii) associativity
$a\ t\ (b\ t\ c) = (a\ t\ b)\ t\ c;$ (2.10)

(iii) boundary conditions:
$a\ t\ 0 = 0, \quad a\ t\ 1 = a;$ (2.11)

(iv) monotonicity
for $a_1<a_2$ and $b_1<b_2$ then
$a_1\ t\ b_1 \leq a_2\ t\ b_2$ (2.12)

where $a, b, a_1, b_1, a_2, b_2, c \in [0, 1]$.

Similarly, by s-norm, we mean a two place function which preserves the following conditions:

$$s:[0,1]\times[0,1] \rightarrow [0,1]$$

(i) commutativity:
$a\ s\ b = b\ s\ a;$ (2.13)

(ii) associativity
$a\ s\ (b\ s\ c) = (a\ s\ b)\ s\ c;$ (2.14)

(iii) boundary conditions:
$a \, s \, 0 = a, \quad a \, s \, 1 = 1;$ (2.15)

(iv) monotonicity
for $a_1 < a_2$ and $b_1 < b_2$ then
$a_1 \, s \, b_1 \leq a_2 \, s \, b_2$ (2.16)

where $a, b, a_1, b_1, a_2, b_2, c \in [0, 1]$.

Table 2.1 shows various examples of t-norms and s-norms [16]. The t-norms refer to the models of intersection of fuzzy sets, while s-norms are models of union of fuzzy sets. Also t-norms and s-norms are related by the De Morgan law,

$$a \, s \, b = 1 - (1 - a) \, t \, (1 - b) \tag{2.17}$$

This relationship states that for any t-norm, one can find its dual. All the t-norms and s-norms given in Table 2.1 satisfy this relationship.

t-norms	s-norms
$a \, t_1 \, b = Min[a, b]$	$a \, s_1 \, b = Max[a, b]$
$a \, t_2 \, b = a \cdot b$	$a \, s_2 \, b = a + b - a \cdot b$
$a \, t_3 \, b = \dfrac{a \cdot b}{\gamma + (1 - \gamma) \cdot [a + b - a \cdot b]}, \gamma \geq 0$	$a \, s_3 \, b = \dfrac{a \cdot b \cdot (\gamma - 2) + a + b}{a \cdot b \cdot (\gamma - 1) + 1}, \gamma \geq 0$
$a \, t_4 \, b = Max[0, (\lambda + 1) \cdot (a + b - 1) - \lambda \cdot a \cdot b], \lambda \geq -1$	$a \, s_4 \, b = Min[1, a + b + \lambda \cdot a \cdot b], \lambda \geq -1$

Table 2.1 Examples of Triangular Norms.

2.4 Artificial Neural Network

In the last three decades, computers have proven to be a useful tool in scientific research. The speed of the computational power of computer has been increased by orders of magnitude in the last ten years. But still, there are some tasks like recognizing continuous speech and real-time visual processing that the traditional computer can not perform satisfactorily.

Everyday observation shows that the modest brain of lower animals can perform tasks that are far beyond the range of even the largest and fastest modern computers. This is especially true for those tasks that require the recognition of complex optical or acoustical patterns, which are not determined by simpler logical rules. This shifted the interest of some researchers to focus on investigating and developing the brain-like machine known as Artificial Neural Network (ANN).

An ANN is inspired by the structure of the nerve cells, and their interconnections within the human brain. Instead of a step by step procedure for carrying out the desired associations (or mappings) between objects in response to their environments, the ANN itself generates its own internal rules governing the associations, and refines the rules by comparing the results to the examples (or training sets). Through trial and error, the network literary teaches itself to carry out the task.

2.4.1 Background of ANN

The history of ANN can be tracked back to the 1940s, when Warren McCulloch and Walter Pitts [12] proposed a general theory of information processing based on networks of binary switching (or decision) elements, which are somewhat called 'neurons'. Then a number of theories began to evolve regarding representations of biological neural connection systems in mathematics. In the past decade, especially after the discovery of the well-known error backpropagation algorithm, ANN has led to a new resurgence of the field.

In general, ANN is a structured network of many very simple processors (neurons), which are connected by unidirectional communication channels (as synapses in the brain) to carry data. The processing elements in ANN are modeled from the structure of the brain nerve cells (or neurons), for which they receive the stimuli from some other neurons. Some of these connections between the neurons are strong, and some of them are weak. The neuron accepts the inputs, and generates a response to other neurons. Figure 2.1 shows the simplified structure of neurons and their connections.

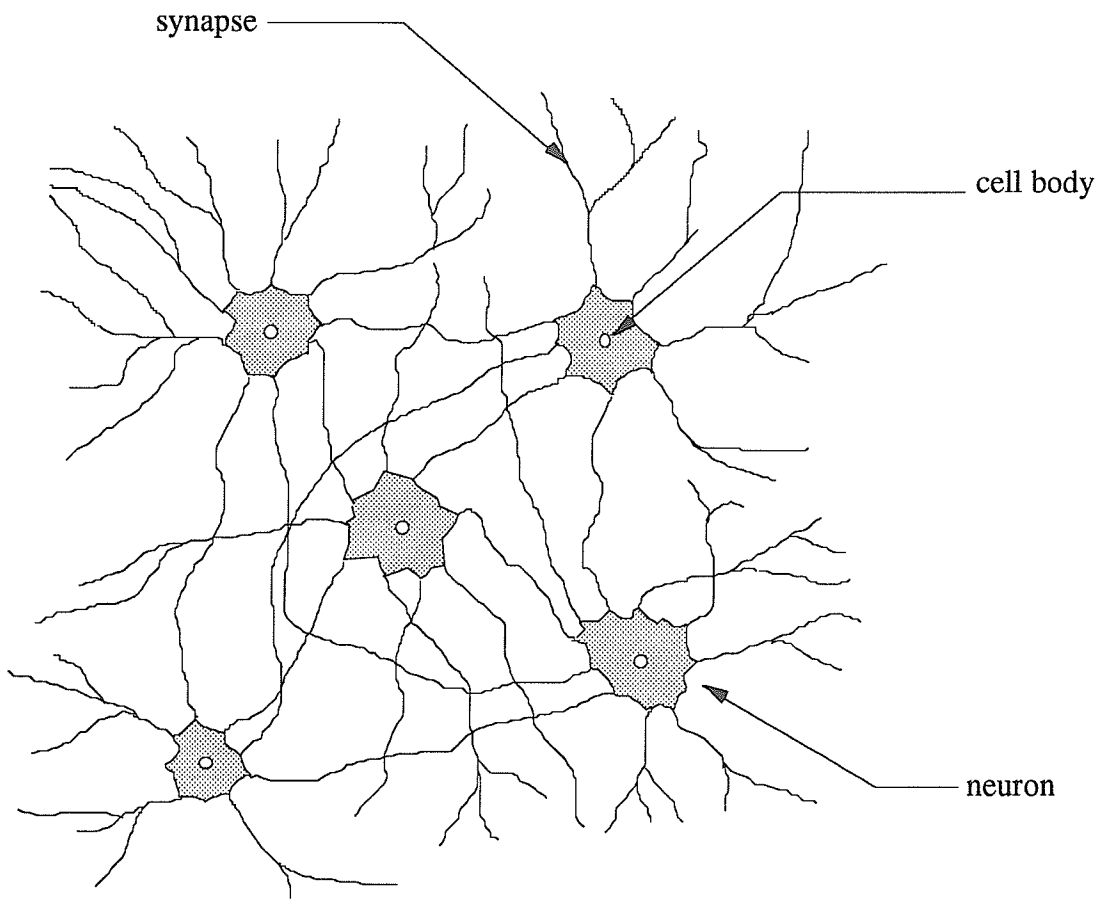


Figure 2.1 Simplified Structure of Neurons and Their Connections.

A typical processing element in ANN is shown in figure 2.2. Signal x_i is the input to the processing element (or artificial neuron), θ is the bias to the processing element, and w_i is the connection strength (or weight) from the input x_i to the processing element. y is the output of summing the product of x_i and w_i for all the incoming signals. The output y is then passed to a transfer function that determines the processing element's

output signal as a function of the most recent input signals and the weights. The final output of the processing element is represented by y_{out}. The transfer function provides the non-linearity constraint at the output of the processing element.

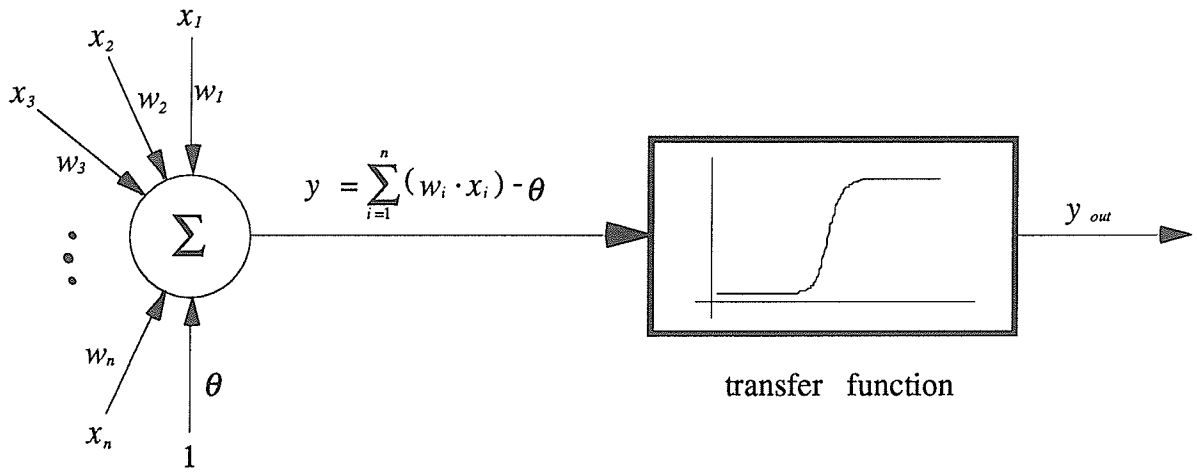


Figure 2.2 Typical Processing Element in the ANN.

A typical ANN consists of many processing elements. Each processing element has many inputs, but only one single output which is fanned out along many pathways to provide inputs to other processing elements. These pathways are interconnected with the processing elements to form a network. Figure 2.3 shows a typical ANN that consists of one input layer, one to several hidden layers, and one single output layer. The processing elements in one layer receive outputs from the previous (or upper) layer, process them with the other processing elements in parallel, and deliver outputs to the processing elements in the next (or lower) layer. Since the inputs and the connections in the network are modifiable over time, the network can learn.

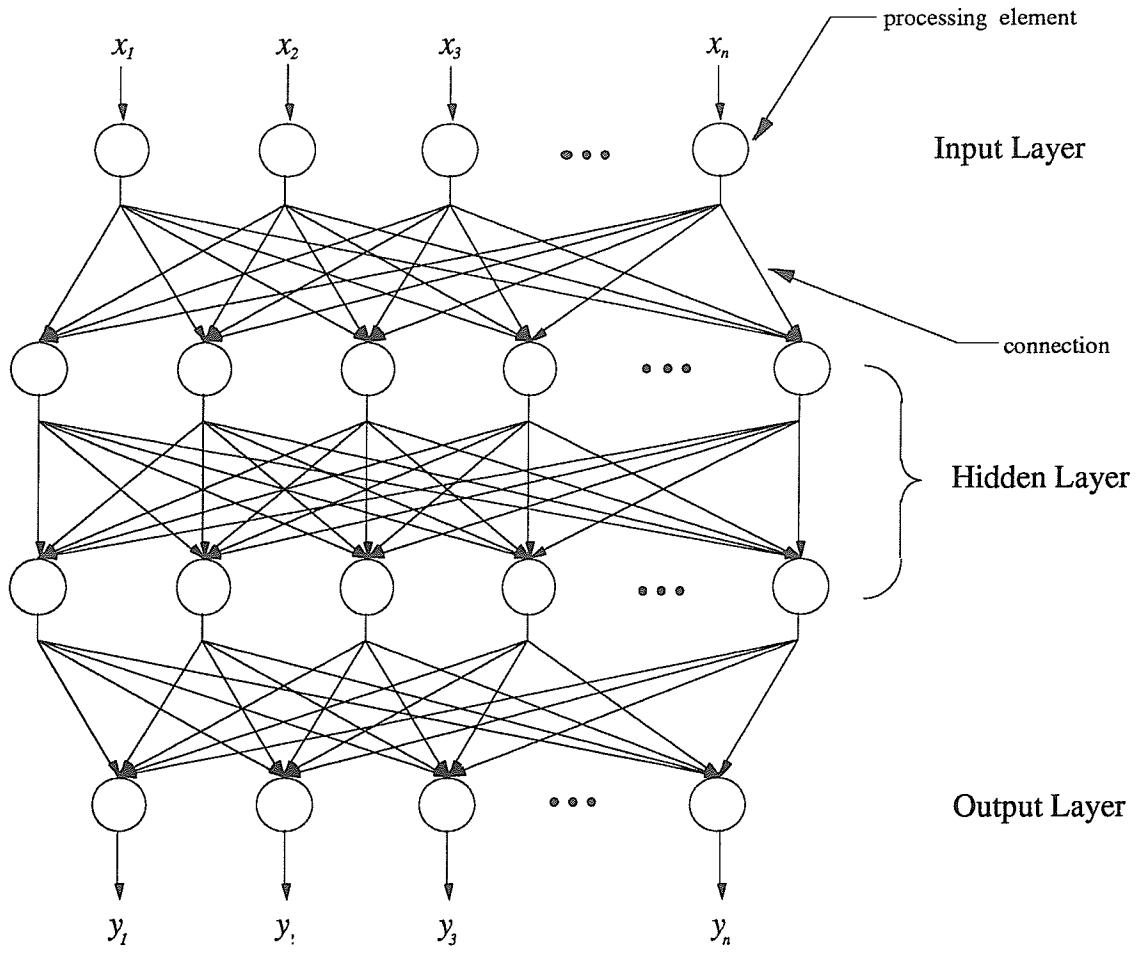


Figure 2.3 Typical Artificial Neural Network.

2.4.2 Varieties of Learning Procedure of ANNs

The existing ANN models with different learning algorithms can be categorized into two groups based on the learning algorithms: supervised learning and unsupervised

learning. In supervised learning, we know the desired outputs for each input vector presented to the ANN. This allows us to compute an error produced by the network that is based on the actual output and the desired output for each training pattern. This error is then used to guide the learning process of the ANN.

In unsupervised learning, the ANN is autonomous. The network simply looks at the presented input data, finds out some of the global properties there and learns to accommodate these properties in its weights. No desired output vectors are given to the network. In general, the tasks of unsupervised learning are usually more challenging than the problems of supervised learning. They are particularly useful in cases in which we do not possess complete information about the data environment.

2.4.3 Topology of ANNs

We can distinguish two basic types of ANNs considering their topologies. This gives rise to recurrent nets and feedforward nets. In recurrent nets, the elements in the network are fully connected, and the processing elements are updated episodically by a scheduling function. In feedforward nets, the network topology is defined into layers. The processing elements in layer i accept the output from layer i-1 as inputs, calculate the corresponding outputs in parallel, and feed them to the successive (i+1)-th layer.

2.5 Backpropagation Algorithm

The backpropagation algorithm was first described by P. Werbos [25] in 1974, but did not becoming popular until it was researched thoroughly by D. E. Rumellhart, G. E. Hinton and R. J. Williams [22], and simultaneously by two other researchers Le Cun [8] and Parker [13]. Backpropagation uses a gradient search technique to minimize the mean square error between the desired and the actual network outputs. The implementation of backpropagation involves a forward pass through the layers to estimate the error, and a backward pass modifying the connection strengths (weights) to decrease the error.

Backpropagation applies the generalized delta rule to adjust the weights of the processing elements [9]. Adjustments are made to the weights in the processing elements in proportion to the amount on that weight contributed to the error.

Table 2.2 is a summary of the backpropagation training algorithm [9], with the assumption that the transfer function in the processing element is a sigmoid logistic function $f(x)$

$$f(x) = \frac{1}{1+e^{-x}} \tag{2.18}$$

The sum of mean square errors e_p for the training set is defined as

$$e_p = \frac{1}{2}\sum_{k=1}^{n}\left(y_{pk} - d_{pk}\right)^2 \tag{2.19}$$

where d_{pk} is the desired output of processing element k at the presentation of training pattern p, y_{pk} is the actual output for processing element k at the output layer under the presentation of training set pattern p; n is the total number of patterns in the training set.

Step 1	Initialize the weights in the network by setting them to some random values.
Step 2	Present the input vector and the desired output vector from the training set to the network.
Step 3	Use the sigmoid function in equation (2.18) and formula as in figure 2.2 to calculate the actual output vector. Then use the mean square error equation presented in (2.19) to measure the error.
Step 4	Use a recursive algorithm starting at the output nodes and working backward to the first hidden layer. The weight is updated by $$w_{ij}(t+1) = w_{ij}(t) + \eta \cdot \delta_j \cdot y_i'$$ in the above equation, $w_{ij}(t)$ is the weight in the connection from processing element j to processing element i at time t, η is the learning rate of the adaptive rule, δ_j is an error term for processing element j, y_i' is the output of node i, or the input at node i if it is at the input layer. For processing element j in the output layer, the value of δ_j is equal to $$\delta_j = y_{pj} \cdot (1 - y_{pj}) \cdot (d_{pj} - y_{pj})$$ or if processing element j is located in the hidden layer, then δ_j is equal to $$\delta_j = y_j' \cdot (1 - y_j') \cdot \sum_{k=1}^{h} (\delta_k \cdot w_{jk})$$ where y_{pj} is the output of processing element j at the output layer under the presentation of pattern p in the learning set; d_{pj} is the desired output in the presentation of pattern p in the learning set; h is the total number of processing element in the next hidden layer.
Step 5	Repeat the process by going to step 2 until some stopping conditions are met.

Table 2.2 The Backpropagation Algorithm.

2.6 Why Neural Network

At the present state of the development of neural networks, the traditional computers are still superior to neural networks in performing the tasks whose structure has been thoroughly analyzed. However, it is only fair to state that many years of analysis are required to understand the problem well enough to be accessible to computational treatment. On the other hand, a neural network may be able to learn from a large number of examples to adjust the correct weightings in just a few days. Moreover, since the neural network operates in parallel, once the correct connections between the processing elements are established, the execution may be much faster than on the traditional computer.

Another important aspect is the potential capability of generalization and classification in neural networks. Although the traditional AI can solve these tasks if the classification criteria or the generalization rules are known, they are not good at establishing such relationships. This is different in neural networks, where they can learn and generalize from examples without knowing any precise logical description in advance. Only an informal understanding of the complexities of the desired behavior is sufficient to construct the overall architecture of an appropriate neural network.

Neural networks store the captured knowledge into the connections between the neurons with distributed encoding, which endows the network with fault tolerance. In the traditional computer, a failure in one single instruction or data may cause the computer to fail completely in its operation. However, the operation of a neural network often remains unaffected if a single neuron fails or a few connections break down.

Of course, neural networks possess some disadvantages When we look inside the neural network, we lack an understanding of how the neural network actually has solved a given problem. It is hard to determine whether the results provided by the neural network are correct, especially when the logical structure of the solution for the problem is not known. Also, there is no way to judge the performance of a neural network by knowing only its architecture, since it is almost impossible to determine what tasks that the network can actually perform from its stored knowledge.

2.7 Fuzzy Neural Network

The formal structure of fuzzy sets and their operators can be viewed as an interesting tool in mapping the logic properties of some problems into the topologies of artificial neural networks [17]. The logic operators of fuzzy sets can serve as a mapping of data in the unit hypercube, and can be implemented as the processing elements of ANN. When put together, they form a new class of neural network - Fuzzy Neural Network (FNN).

Fuzzy neural networks exploit t- and s-norms (see section 2.3.2) to implement the new logic-oriented models of neurons. These logic-oriented neurons can be classified into three different categories: the *OR* neuron, *AND* neuron (aggregative neurons) and the *Reference* neuron. Each of these neurons models different logic sets connectives. The following sections will discuss them in more detailed.

2.7.1 OR neuron

The *OR* neuron realizes a logical union of the input signals. Figure 2.4 shows an *OR* neuron, with a n-dimensional input signal $x = [x_1, x_2, ..., x_n]^T \in [0, 1]^n$, and the weights (or connections) $w = [w_1, w_2, ... w_n]^T \in [0, 1]^n$. The *OR* neuron is governed by the formula:

$$y = OR\ (x, w) \tag{2.20}$$

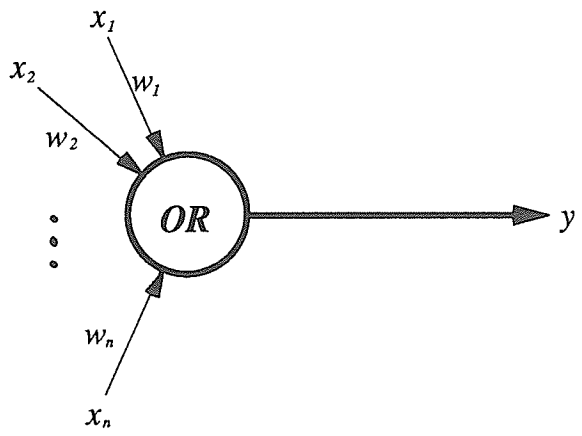


Figure 2.4 A Fuzzy OR Neuron.

The *OR* neuron uses s-t composition operator. The input signal x_i is affected by the connection w_i,

$$(x_i\ t\ w_i) \triangleq (x_i\ AND\ w_i) \tag{2.21}$$

The role of the connection w_i is to modulate the influence of the input signal x_i on the final result of aggregation y. The partial result of the term $(x_i\ t\ w_i)$ is combined with other weighted input signals through the s-norm (OR) operator. Now expression (2.20) becomes

$$\begin{aligned} y &= \mathop{\mathbf{S}}_{i=1}^{n}(x_i\ t\ w_i) \\ &= (x_1\ AND\ w_1)\ OR\ (x_2\ AND\ w_2)\ OR \cdots OR\ (x_n\ AND\ w_n) \end{aligned} \tag{2.22}$$

In the limit case of $w_j = 1$, one gets

$$\begin{aligned} y &= \mathop{\mathbf{S}}_{i=1}^{n}(x_i\ t\ w_i) \\ &= \left[\mathop{\mathbf{S}}_{i=1,i\neq j}^{n}(x_i\ t\ w_i)\right] s\,(x_j\ t\ 1) \\ &= \left[\mathop{\mathbf{S}}_{i=1,i\neq j}^{n}(x_i\ t\ w_i)\right] s\,(x_j) \end{aligned} \tag{2.23}$$

that is, no modification is made to the input signals x_j through the weight w_j. If $w_j = 0$, we have

$$\begin{aligned} y &= \mathop{\mathbf{S}}_{i=1}^{n}(x_i\ t\ w_i) \\ &= \left[\mathop{\mathbf{S}}_{i=1,i\neq j}^{n}(x_i\ t\ w_i)\right] s\,(x_j\ t\ 0) \\ &= \left[\mathop{\mathbf{S}}_{i=1,i\neq j}^{n}(x_i\ t\ w_i)\right] s\,(0) \\ &= \left[\mathop{\mathbf{S}}_{i=1,i\neq j}^{n}(x_i\ t\ w_i)\right] \end{aligned} \tag{2.24}$$

and the input signal x_j becomes completely eliminated and makes no contribution to the result y.

2.7.2 AND Neuron

The structure of the *AND* neuron is similar to the *OR* neuron, however the operators (t- and s- norms) are utilized in a reversed orders. An *AND* neuron realizes an intersection of the input signals. Figure 2.5 shows this neuron, with a n-dimensional input signal $x = [x_1, x_2, ..., x_n]^T \in [0, 1]^n$, and the weights (or connections) $w = [w_1, w_2, ... w_n]^T \in [0, 1]^n$. The *AND neuron* is governed by the formula:

$$y = AND\ (x, w) \tag{2.25}$$

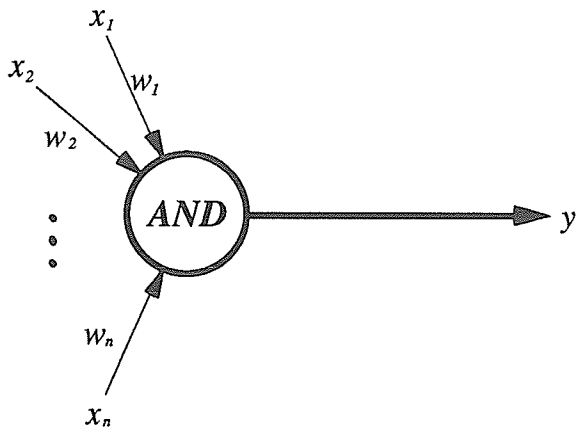


Figure 2.5 A Fuzzy AND Neuron.

The *AND* neuron is implemented with t-s composition operator. The input signal x_i is weighted by the connection w_i through the s-norm (*OR*) operator:

$$(x_i \; s \; w_i) \triangleq (x_i \; OR \; w_i) \tag{2.26}$$

and then this partial information is combined with other weighted input signals by the t-norm (*AND*) operator, which yields an overall *AND* combination. The expression (2.25) becomes

$$\begin{aligned} y &= \prod_{i=1}^{n} (x_i \; s \; w_i) \\ &= (x_1 \; OR \; w_1) \; AND \; (x_2 \; OR \; w_2) \; AND \cdots AND \; (x_n \; OR \; w_n) \end{aligned} \tag{2.27}$$

In the limit case of $w_j = 1$, the term $(x_j \; s \; w_j)$ in the above equation is equal to 1. Subsequently, x_j has no impact on y. On the other hand, if $w_j = 0$, no modification is made to the input signal x_j.

2.7.3 Inhibitory Behavior of the Fuzzy Neurons

We have shown the performance and characteristics of *OR* and *AND* neurons in the previous two sections. Due to monotonicity of the t- and s-norms, the inputs of these neurons are always excitatory: the higher the value of x_i, the higher the output value of y. Now we can discuss the way to incorporate the inhibitory behavior of the input signals to the *OR* and *AND* neurons.

It is well known that some of the boolean functions are partially dependent on the inhibitory behavior of the input signals. Inhibitory signals can be incorporated in our *OR* and *AND* neurons by admitting complements of x_i as additional inputs.

Recall that the complement of x_i is obtained vice a straightforward formula:

$$\overline{x_i} = 1 - x_i \qquad \text{for} \quad x_i \in [0,1] \tag{2.28}$$

Now when x_i increases, $\overline{x_i}$ decreases and in sequel y decrease as well. Figure 2.6 & 2.7 show the *OR* and *AND* neurons with the complemented input included.

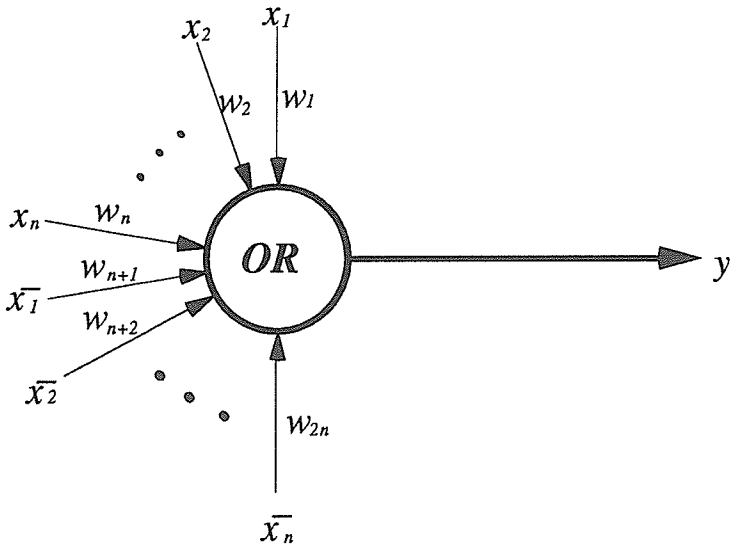


Figure 2.6 A Fuzzy OR Neuron with Complemented Inputs.

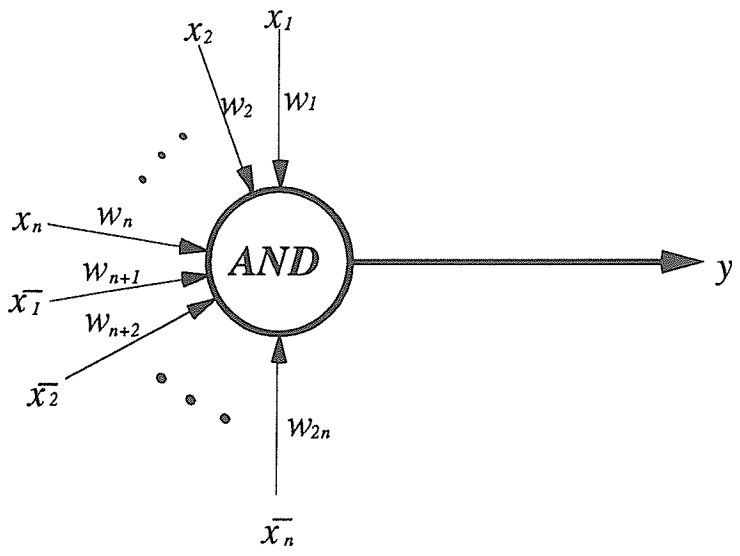


Figure 2.7 A Fuzzy AND Neuron with Complemented Inputs.

2.7.4 Reference Neuron

The final basic type of fuzzy neurons in the fuzzy neural network is the *REF* (REFerence) neuron [17]. The *REF* neurons can perform different tasks like expressing equality, inclusion, dominance and difference operations regarded to an associated reference vector. For simplicity, we will only discuss here the REF_{equ} (equality reference) neuron in details. The reader can refer to [18][19] for more information on the other types of reference neurons.

The REF_{equ} neuron realizes matching of a n-dimensional input signal $x = [x_1, x_2, ..., x_n]^T \in [0, 1]^n$, with regard to an associated reference vector $r = [r_1, r_2, ..., r_n]^T \in [0, 1]^n$. The REF_{equ} neuron is governed by the formula

$$y = w \circ (x \equiv r) \tag{2.29}$$

with $y \in [0, 1]$. The equality operator $\equiv$ is defined using a pseudocomplement $\rightarrow$ associated with a given t-norm:

$$a \equiv b \;=\; \frac{1}{2}\left[(a \rightarrow b) \wedge (b \rightarrow a) + (\bar{a} \rightarrow \bar{b}) \wedge (\bar{b} \rightarrow \bar{a})\right] \tag{2.30}$$

with $\bar{a}$ denotes the negation of a, in which $\bar{a} = 1 - a$; $a \rightarrow b = sup\ \{ c \in [0, 1] \mid a\ t\ c \leq b\}$, and $a \wedge b = min(a, b)$; for all $a, b \in [0, 1]$.

The neuron uses the s-t composition operator "$\circ$" and aggregates the results of matching $x \equiv r$, with the weight w_i that modulate the influence of each individual term. Expression (2.29) can be rewritten as

$$\begin{aligned} y &= \mathop{S}_{i=1}^{n} w_i\ t\left(x_i \equiv r_i\right) \\ &= \left[w_1\ AND\left(x_1 \equiv r_1\right)\right] OR \left[w_2\ AND\left(x_2 \equiv r_2\right)\right] OR \cdots OR \left[w_n\ AND\left(x_n \equiv r_n\right)\right] \end{aligned} \tag{2.31}$$

In the limit case of $w_j = 1$, one gets

$$\begin{aligned} y &= \mathop{S}_{i=1}^{n} w_i\ t\left(x_i \equiv r_i\right) \\ &= \left[\mathop{S}_{i=1, i\neq j}^{n} w_i\ t\left(x_i \equiv r_i\right)\right] s\left(1\ t\left(x_j \equiv r_j\right)\right) \\ &= \left[\mathop{S}_{i=1, i\neq j}^{n} w_i\ t\left(x_i \equiv r_i\right)\right] s\left(x_j \equiv r_j\right) \end{aligned} \tag{2.32}$$

Moreover, if $x_j = r_j$, the result of y in (2.32) equals 1. If $w_j = 0$, we have

$$\begin{aligned} y &= \mathop{\mathbb{S}}_{i=1}^{n} w_i \, t\left(x_i \equiv r_i\right) \\ &= \left[\mathop{\mathbb{S}}_{i=1, i\neq j}^{n} w_i \, t\left(x_i \equiv r_i\right)\right] s\left(0 \, t\left(x_j \equiv r_j\right)\right) \\ &= \mathop{\mathbb{S}}_{i=1, i\neq j}^{n} w_i \, t\left(x_i \equiv r_i\right) \end{aligned} \tag{2.33}$$

and the influence of the j-th term in (2.31) is completely eliminated, no matter what the values for x_j and r_j are.

When all the input signals to the REF_{equ} neuron are treated uniformly (with all w_i = 1), then expression (2.31) becomes

$$y = \mathop{\mathbb{S}}_{i=1}^{n} \left(x_i \equiv r_i\right) \tag{2.34}$$

Furthermore, if for some x_i has an ideal matching with r_i, then the output y of the REF_{equ} neuron equals to 1.

2.8 Architecture of the Fuzzy Neural Network

As we know in the boolean logic, any arbitrary function can be expressed by the Sum of Minterms (SOM) or its dual, the Product of Maxterms (POM). Therefore, any digital circuit can be constructed by the *OR* gates, *AND* gates and the inverters provided with no restriction on the number of components being used [11]. Consider the fuzzy logic as an extension to the boolean logic, and recall that the *OR* neuron realizes the

union of the inputs, the *AND* neuron realizes the intersection of the inputs. We can apply the same concept to use the *OR* and *AND* neurons, and the complement of the input signals to approximate any function in a logical fashion. This suggests directly two possible topologies of the fuzzy neural network (FNN) to be exploited. The first network will perform the SOM-type of approximation while the second one looks for the POM-type form of approximations [18].

2.8.1 Fuzzy Neural Network in SOM Form

In the first approach, we construct the FNN to produce the SOM form and call it a *AND-OR* fuzzy neural network. Such a network consists of three layers and is fully connected. The input layer consists of $2n$ nodes which includes all n inputs and their complements. The role of the input layer is to distribute the input signals to all the *AND* neurons in the hidden layer. The hidden layer consists of *'h' AND* neurons, and its function is to construct the successive minterms of the input signals. The output layer has only a single *OR* neuron, and is used to combine (through a logical union) all the outputs from the hidden layer. Figure 2.8 shows an *AND-OR* fuzzy network.

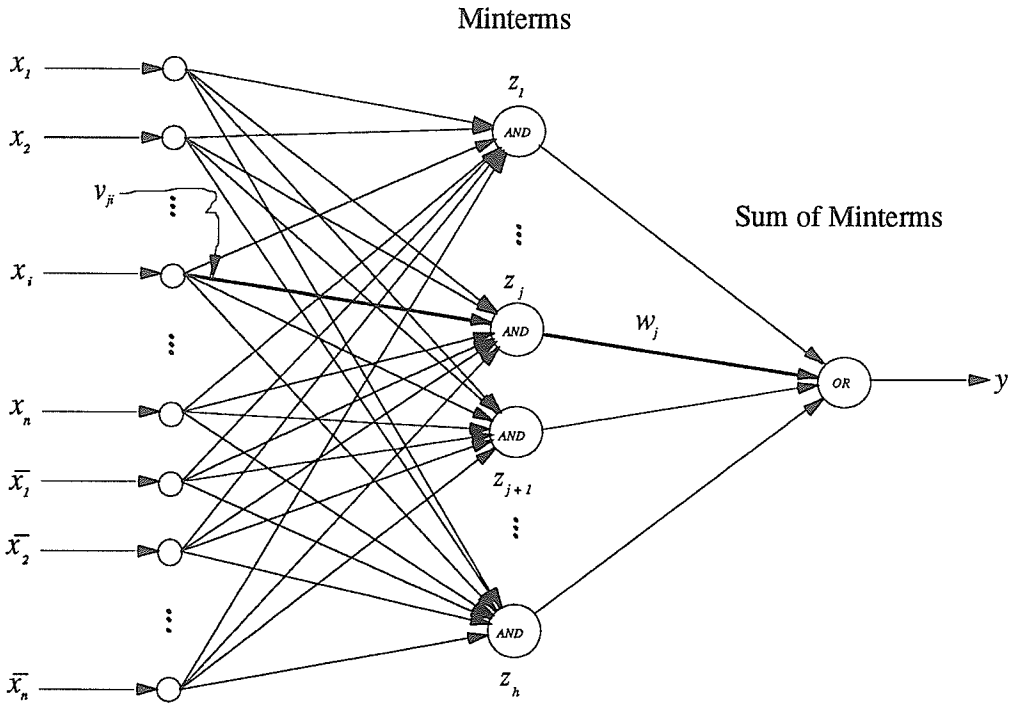


Figure 2.8 An *AND-OR* Fuzzy Neural Network.

Let us assume that the connections between the node i in the input layer to the *AND* neuron j in the hidden layer be v_{ji}, and the connections between the *AND* neuron j in the hidden layer to the *OR* neuron in the output layer denoted by w_j as shown in figure 2.8. Then, the network is governed by the following expressions (see also figure 2.8):

output-hidden Layer

$$y = \sum_{j=1}^{h} \left(w_j \; t \; z_j \right) \tag{2.35}$$

hidden-input layer

$$z_j = \prod_{i=1}^{2n} \left(v_{ji} \, s \, x_i \right) \tag{2.36}$$

This network is heterogeneous, with each layer composed of different types of neurons.

2.8.2 Fuzzy Neural Network in POM Form

The architecture of the fuzzy neural network can also be constructed following a dual approach, in which a function is approximated in its POM version . The structure of the network is similar to the SOM form, and is shown in figure 2.9. Now, the network consists of three layers and is fully connected. The function of the input layer is to distribute the input signal to all the *OR* neurons in the hidden layer. The *OR* neurons in the hidden layer is used to combine the input signals form the input layer in order to construct the maxterms. The output layer consists of a single *AND* neuron, which combines the maxterms generated from the hidden layer. We call this an *OR-AND* fuzzy neural network.

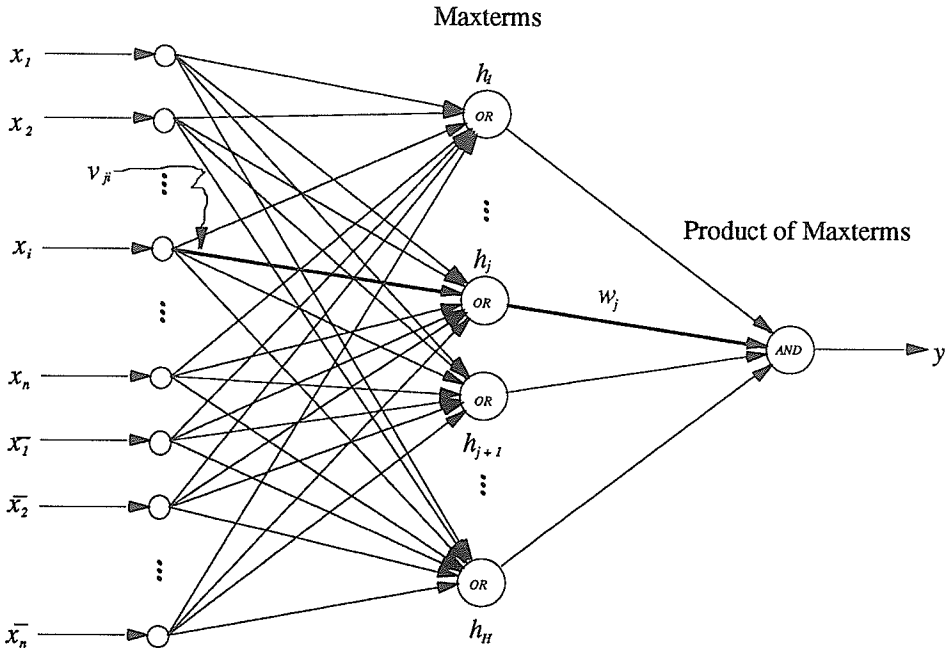


Figure 2.9 An *OR-AND* Fuzzy Neural Network.

Using the same notations as in the *AND-OR* fuzzy neural network, the *OR-AND* fuzzy neural network can be described accordingly

output-hidden layer

$$y = \prod_{j=1}^{h} \left(w_j \; s \; z_j \right) \tag{2.37}$$

hidden-input layer

$$z_j = \sum_{i=1}^{2n} \left(v_{ji} \; t \; x_i \right) \tag{2.38}$$

2.9 Conclusions

The background of fuzzy sets theory, artificial neural network and fuzzy neural network were given. The collection of new logic-oriented models of neurons has been introduced. Detailed discussion on the learning algorithms with the *OR-AND* fuzzy neural network will be given in Chapter 3. Those who are interested in pursuing studies in artificial neural networks and fuzzy systems may refer to [7].

Chapter 3
Logic Processor

3.1 Logic Processor

The Logic Processor (LP) is a new class of fuzzy neural network with a strong logical structure, and is constructed from an *AND-OR* three-layered fuzzy neural network described in section 2.8.1. The structure of the LP is organized to approximate the logical relationships between the inputs vector $i = [i_1, i_2, ..., i_n]^T$ and the output y efficiently. The input layer of the LP consists of $2n$ input nodes, which are for distributing the inputs $i_1, ..., i_n$, as well as their complements $\bar{i}_1, \ldots, \bar{i}_n$. to the hidden layer. The hidden layer consists of h *AND* neurons, and the output layer comprises of an *OR* neuron.

The output y of the LP can be expressed by

$$y = \mathop{\mathrm{S}}_{j=1}^{h} \left(w_j \ t \ z_j \right) \tag{3.1}$$

where w_j is the strength of the connection between element j in the hidden layer to the output element. z_j is the output from the element j in the hidden layer, and is obtained by

$$z_j = \mathop{\mathrm{T}}_{b=1}^{2n} \left(v_{jb} \ s \ x_b \right) \tag{3.2}$$

v_{jb} denoting the connection between the node b in the input layer to the element j in the hidden layer; x_i represents the i^t-th input in the input layer.

The primary aim of the chapter is to consider learning in this class of the logic-oriented network. We will first look at the performance indices guiding the learning procedure and afterwards concentrate on the non-parametric and parametric learning.

3.2 Performance Indices

The overall performance of the logic processor is evaluated by the performance index (Q). Two different performance indices were used throughout the research, namely the equality index Q_1 and the sum of square errors Q_2. The objective of the performance index is to monitor the performance of the LP in the course of learning.

3.2.1 Equality Index

The equality index Q_1 is first described in [15] by W. Pedrycz, and is given in the following form

$$Q_1 = (a \equiv b) = \frac{1}{2}\left[(a \rightarrow b) \wedge (b \rightarrow a) + (\overline{a} \rightarrow \overline{b}) \wedge (\overline{b} \rightarrow \overline{a})\right] \qquad (3.3)$$

where a and $b \in [0, 1]$.

The above definition was described in section 2.7.4 briefly, and we will discuss it in more details here. This definition has a strong logical background. The '$\rightarrow$' sign

denotes an implication between the two objects a and b. The first component $(a \rightarrow b) \wedge (b \rightarrow a)$ is viewed as a minimum degree to which a implies b, and vice versa. The second component $(\bar{a} \rightarrow \bar{b}) \wedge (\bar{b} \rightarrow \bar{a})$ preserves the same structure as the first one. The only difference is that it applies to the complements of the values a and b. In contrast to two-valued logic where these two components coincide, their multivalued versions produce different results. Therefore, it is prudent to average these partial results as indicated in (3.3)

Among many models of implications, the one proposed by Lukasiewicz is particularly attractive due to its simple realization.

$$a \rightarrow b \quad = \quad \begin{cases} 1+b-a, & \text{if } a > b \\ 1, & \text{otherwise} \end{cases} \tag{3.4}$$

Inserting (3.4) into (3.3), we derived the following expression for the equality index:

$$a \equiv b \quad = \quad \begin{cases} 1+a-b, & \text{if } b > a \\ 1+b-a, & \text{if } b < a \\ 1, & \text{if } b = a \end{cases} \tag{3.5}$$

Thus a ≡ b becomes a piecewise linear function of its arguments. Considering the training set consisting of n elements (pairs (x_k, d_k)), the performance index Q_1, to be maximized by a LP is defined as

$$Q_1 = \sum_{k=1}^{n} (d_k \equiv y_k) \tag{3.6}$$

where d_k denotes the desired output for the k-th pattern in the training set, while y_k represents the actual output form the network (LP) for pattern k.

3.2.2 Mean Square Error

The mean square error Q_2 that is used to measure the performance of the LP is given by

$$Q_2 = \frac{1}{2}(y-d)^2 \tag{3.7}$$

with d standing for the desired value of the output, and y is the actual output.

For a given training set with n patterns, the overall performance index Q_2 will be taken as a sum of errors

$$Q_2 = \frac{1}{2}\sum_{k=1}^{n}(y_k - d_k)^2 \tag{3.8}$$

where d_k denotes the desired output for pattern k of the training set, whereas y_k is the actual output of the network for the k-th pattern in the training set.

3.3 Learning Rule

The learning rule developed here is based on the backpropagation algorithm that has been discussed in the previous chapter. The learning procedure of the LP can be classified into two categories, namely non-parametric learning and parametric learning.

Non-parametric learning is concerned with the structure of the LP, such as the number of units in the hidden layer, and specification of the t- and s-norms used in the *OR* neuron and *AND* neuron. On the other hand, parametric learning is concerned with the numerical changes of the connections. Our objective for this learning of the LP is to adjust the connections in such a way that the LP can obtain the best possible logical mapping between the input and output spaces. With this mapping at hand, we can then examine the connections inside the LP and find out the logical relationships between the data.

3.3.1 Non-parametric Learning

The non-parametric learning includes the factors which are outside the realm of the standard gradient-oriented optimization techniques used in the backpropagation, such as the number of units in the hidden layer, and the choice of triangular norms.

Hidden Units

The number of units h in the hidden layer represents the number of generalized minterms that the network can recognize. In general, we want to keep this number h as low as possible to provide a compact realization of the logical relationships of the system. However, we also want to minimize the error in the LP and this requires a certain minimal number of units in the hidden layer to capture the basic logical relationships underlying the system. As we can see, there may exist a trade off between these two criterion, and we want to satisfy both criterion as much as possible. One way to solve the

problem is to start with a minimal number of units h in the hidden layer (like 1), and increase this number h until the performance index is less than an error ε, where ε is a very small non-negative constant. Or else, we can work in another way. We will start with a maximum number of units h in the hidden layer[1] , say $2n$ (where n is the total number of input to the LP), and decrement the number of units h until some criterion are satisfied.

Selection of Triangular Norms

The choices of the triangular norms do not pose too many constraints, but we should keep in mind that the different triangular norms imply different levels of binding of the arguments. For example, the Max operator of s-norm is completely non-interactive,

$$Max(a,x)=\begin{cases} x & if \quad a<x, \\ a & if \quad a\geq x. \end{cases} \tag{3.9}$$

that means, the result of this operator remains unchanged for all the values of a greater than or equal to x.

However, the Probabilistic Sum (PS) viewed as another example of s-norm is interactive,

$$PS(a,x)=a+x-a\cdot x \tag{3.10}$$

1 In boolean algebra, the maximum number of minterms is $2n$, where n is the total number of inputs. Since LP is a network tries to approximate a function in the SOM form, we make the LP carries a maximum of $2n$ units in the hidden layer.

that is, the result of this operator is dependent on both the arguments.

As a result of the above observation, we can conclude that the selection of the triangular norms should be application oriented. We should assume or roughly estimate the interactivity between the variables, and pick a suitable triangular norms.

3.3.2 Parametric Learning

The changes of connections in the LP are defined as proportional to the derivative of the performance index Q with respect to the connections. Putting this in symbolic form, it becomes

$$\Delta connection(t) = -\frac{\partial Q}{\partial\, connection(t)} \tag{3.11}$$

and the connections in the LP are updated iteratively accounting to:

$$connection(t+1) = connection(t) + \Delta connection(t) \tag{3.12}$$

If we change each of the connections in the LP according to (3.12), then the weight in each of the connections is moving toward the minimum of the performance index (using mean square error Q_2). This corresponds to a system as moving downhill in the error surface of the weight space[1] until it reaches its minimum point.

1 The height of the error surface at any point in the weight space is equal to the error calculated at these weights.

On line and Off Line Learning

Two techniques are used to find the changes in the connections: on-line learning and off-line learning. In on-line learning, modification of the connections are made after the presentation of an individual pair from the training data set. The performance index is calculated, using the equality index or mean square error,

$$Q_1 = (d_k \equiv y_k)$$
$$or \qquad Q_2 = \frac{1}{2}(y_k - d_k)^2 \tag{3.13}$$

Based on their derivations, the updates take place

hidden-output layer

$$w_c(t+1) = w_c(t) + \eta_{w_c}(t+1)\left[\Delta w_c(t+1) + \alpha \cdot \Delta w_c(t)\right] \qquad c = 1, 2, ..., h \tag{3.14}$$

input-hidden layer

$$v_{ab}(t+1) = v_{ab}(t) + \eta_{v_{ab}}(t+1)\left[\Delta v_{ab}(t+1) + \alpha \cdot \Delta v_{ab}(t)\right] \qquad a = 1, 2, \ldots, h; \quad b = 1, 2, \ldots, m \tag{3.15}$$

for every single training data pair; the input x_k and the output y_k.

The detailed computations of the update rules carried out for Product (P) and Probabilistic Sum (PS) are provided in Appendix A. Table 3.1 summarizes the main results derived for the network with m inputs, and h neurons in the hidden layer.

In off line learning, we accumulate the changes of the connections based on one of the performance indices listed in (3.13) coming from every single training pair x_k and

y_k, and update the connections in the network at the end of the presentation of the entire training set. Table 3.2 includes the corresponding formulas.

The two equations (3.14) & (3.15) shown in the above are derived from (3.12). $\eta_{wc}(t+1)$ is the learning rate for the connection w_c at time $(t+1)$; $\Delta w_c(t+1)$ denotes the changes of connection w_c at time $(t+1)$. The second component inside the brackets of the two equations (3.14) and (3.15) refers to a so-called momentum term, and is controlled by the momentum adjustment parameter α.

Adaptive Learning Rate

A number of mechanisms are being employed here to increase the convergence rate of learning. In this work, a simple and efficient method called adaptive learning rate [24] which alters the learning rate is adopted. The idea of the method is to consider an independent learning rate for each weight in the network. This learning rate is then updated according to

$$\eta_{w_i}(t+1)=\begin{cases} a\cdot\eta_{w_i}(t) & \text{if } \dfrac{\partial Q(t+1)}{\partial w_i} \text{ and } \dfrac{\partial Q(t)}{\partial w_i} \text{ have the same signs.} \\ b\cdot\eta_{w_i}(t) & \text{if } \dfrac{\partial Q(t+1)}{\partial w_i} \text{ and } \dfrac{\partial Q(t)}{\partial w_i} \text{ have opposite signs.} \end{cases} \tag{3.16}$$

where $\eta_{w_i}(t)$ is the learning rate for the connection w_i at time t; $\dfrac{\partial Q(t+1)}{\partial w_i}$ is the derivative of the performance index function at time $(t+1)$ with respect to the connection w_i, a and b are two constants which are slightly above and below 1, respectively.

	Performance Index Q_1
On-line Learning	$\Delta w_c = 0; \quad \Delta v_{ab} = 0 \quad \text{if} \quad y_k = d_k$ $\Delta w_c = \left[\overset{h}{\underset{i=1,i\neq c}{\text{PS}}}\left(w_i \cdot z_i\right) - 1\right] \cdot z_c \quad \text{if} \quad y_k > d_k$ $\Delta w_c = \left[1 - \overset{h}{\underset{i=1,i\neq c}{\text{PS}}}\left(w_i \cdot z_i\right)\right] \cdot z_c \quad \text{if} \quad y_k < d_k$ $\Delta v_{ab} = \left[\overset{h}{\underset{i=1,i\neq a}{\text{PS}}}\left(w_i \cdot z_i\right) - 1\right] \cdot w_a \cdot \left[\overset{m}{\underset{j=1,j\neq b}{\text{P}}}\left(\text{PS}\left(v_{aj} \cdot x_{kj}\right)\right)\right] \cdot \left(1 - x_{kb}\right) \quad \text{if} \quad y_k > d_k$ $\Delta v_{ab} = \left[1 - \overset{h}{\underset{i=1,i\neq a}{\text{PS}}}\left(w_i \cdot z_i\right)\right] \cdot w_a \cdot \left[\overset{m}{\underset{j=1,j\neq b}{\text{P}}}\left(\text{PS}\left(v_{aj} \cdot x_{kj}\right)\right)\right] \cdot \left(1 - x_{kb}\right) \quad \text{if} \quad y_k < d_k$
Off-line Learning	$\Delta w_c = -\sum_{k:y_k > d_k}\left\{\left[1 - \overset{h}{\underset{i=1,i\neq c}{\text{PS}}}\left(w_i \cdot z_i\right)\right] \cdot z_c\right\} + \sum_{k:y_k < d_k}\left\{\left[1 - \overset{h}{\underset{i=1,i\neq c}{\text{PS}}}\left(w_i \cdot z_i\right)\right] \cdot z_c\right\}$ $\Delta v_{ab} = -\sum_{k:y_k > d_k}\left\{\left[1 - \overset{h}{\underset{i=1,i\neq a}{\text{PS}}}\left(w_i \cdot z_i\right)\right] \cdot w_a \cdot \left[\overset{m}{\underset{j=1,j\neq b}{\text{P}}}\left(\text{PS}\left(v_{aj} \cdot x_{kj}\right)\right)\right] \cdot \left(1 - x_{kb}\right)\right\} + \ldots$ $\ldots + \sum_{k:y_k < d_k}\left\{\left[1 - \overset{h}{\underset{i=1,i\neq a}{\text{PS}}}\left(w_i \cdot z_i\right)\right] \cdot w_a \cdot \left[\overset{m}{\underset{j=1,j\neq b}{\text{P}}}\left(\text{PS}\left(v_{aj} \cdot x_{kj}\right)\right)\right] \cdot \left(1 - x_{kb}\right)\right\}$

Table 3.1 Changes of Connections for Different Learning Techniques Using Performance Indices Q_1.

	Performance Index Q_2
On-line Learning	$\Delta w_c = (d_k - y_k)\left[1 - \underset{i=1,i\neq c}{\overset{h}{\mathrm{PS}}}(w_i \cdot z_i)\right] \cdot z_c$ $\Delta v_{ab} = (d_k - y_k)\left[1 - \underset{i=1,i\neq a}{\overset{h}{\mathrm{PS}}}(w_i \cdot z_i)\right] \cdot w_a \cdot \left[\underset{j=1,j\neq b}{\overset{m}{\mathrm{P}}}\left(\mathrm{PS}(v_{aj} \cdot x_{kj})\right)\right] \cdot (1 - x_{kb})$
Off-line Learning	$\Delta w_c = \sum_{k=1}^{n}\left\{(d_k - y_k) \cdot \left[1 - \underset{i=1,i\neq c}{\overset{h}{\mathrm{PS}}}(w_i \cdot z_i)\right] \cdot z_c\right\}$ $\Delta v_{ab} = \sum_{k=1}^{n}\left\{(d_k - y_k) \cdot \left[1 - \underset{i=1,i\neq a}{\overset{h}{\mathrm{PS}}}(w_i \cdot z_i)\right] \cdot w_a \cdot \left[\underset{j=1,j\neq b}{\overset{m}{\mathrm{P}}}\left(\mathrm{PS}(v_{aj} \cdot x_{kj})\right)\right] \cdot (1 - x_{kb})\right\}$

Table 3.2 Changes of Connections for Different Learning Techniques Using Performance Indices Q_2.

Momentum Term

The momentum term $(\alpha \cdot \Delta w_c(t))$ in (3.14) determines the effect of the past weight changes on the current direction of movement in weight space. This momentum term effectively filters out the high frequency variations of the error surface in the weight space [10]. This is especially useful in spaces containing long ravines that are characterized by shape curvature across the ravine and a gently sloping floor, which only allows small changes to prevent oscillations across the ravine. Therefore, the presence of the momentum term will help to filter out the high curvature and thus allows a bigger weight changes step, which leads to a much faster convergence to the equilibrium point in the error surface.

Range of Weights in LP

The learning process that we have just introduced may potentially suffer from the determination of the changes of the connections. Referring to Appendix A, one can observe that if the weight in one of the connections is zero, the derivative of the performance index with respect to this connection will always be zero. This implies that no changes is being made to the connections once their values reach zero, and this may lead to a local lack of convergence. In order to solve the problem, we can force all the weights in the network to narrow down the original operation range from [0, 1] to [*a(t)*, 1-*a(t)*], where *a(t)* is considered be a decreasing function of iteration steps *t*. In the series of simulations reported later, *a(t)* will be expressed as

$$a(t) = \frac{1}{25 + t^{3/2}} \qquad t = 0, 1, 2, \ldots \tag{3.17}$$

3.4 Boolean Approximation

The idea of boolean approximation is to prune meaningless connections (and eventually neurons) to simplify interpretations of the logical relationships residing within the network. Recalling the characteristics of the logic-based neurons by, the following threshold operators are proposed:

For the *AND* neurons,

$$\alpha(w_c) = \begin{cases} 1, & \text{if } w_c > \alpha \\ 0, & \text{otherwise} \end{cases} \tag{3.18}$$

and for the *OR* neurons,

$$\beta(w_c) = \begin{cases} 1, & \text{if } w_c \geq \beta \\ 0, & \text{otherwise} \end{cases} \tag{3.19}$$

In fact, this pruned network gives rise to the boolean form of the logic processor where all the connections are taken from {0, 1}. The problems on how to select the level of α and β should be considered separately.

3.4.1 Determination of the Threshold level α and β

The inputs to the LP lie in the range of 0 to 1, which form a unit hypercube with the dimensions equal to the number of inputs. Thus, a dynamic behavior of the studied system can be described by visualizing a continuous migration of the corresponding points in the unit hypercube. Usually, our input data set is concerned at a certain region within the unit hypercube. Bearing this in mind, the optimal values of α and β are determined based on the algorithm in Table 3.3.

Step 1	Starting with a LP that has been already trained. Present a data set of inputs to the LP, which consists of different combination of inputs x_i ranging from 0 to 1 with a fixed increment step, say 0.1. This data set can be generated in many different ways.
Step 2	Present the new input data set to the LP, generate and record the corresponding desired outputs $\boldsymbol{d}$.
Step 3	Set α and β equal to 0.
Step 4	Using (3.18) and (3.19) to convert the weights into their boolean version. We then calculate Q_1 or Q_2 for the values produced by the LP and its boolean approximation.
Step 5	Increment α and β individually in a fixed step, say 0.1, and repeat step 4 until both the values of equal to 1.
Step 6	The values of α and β producing the lowest(Q_2) or the highest(Q_1) values of the performance index are taken as optimal threshold levels for the boolean version of the LP.

Table 3.3 The Optimal Threshold Level Algorithm.

For example, for a LP with four input, (one state variable x, one input i and their complements $\bar{x}$ and $\bar{i}$), the collection of data (x, i) that is used to determine the optimal threshold levels α and β are of the format shown in Table 3.4.

x	i	$\bar{x}$	$\bar{i}$
0	0	1	1
0	0.1	1	0.9
0	0.2	1	0.8
.			
.			
.			
0	1	1	0
0.1	0	0.9	1
0.1	0.1	0.9	0.9
0.1	0.2	0.9	0.8
.			
.			
.			
1	0.8	0	0.2
1	0.9	0	0.1
1	1	0	0

Table 3.4 Example of Data Used to Determine the Threshold Level for a Four Input LP.

3.5 Induction of the 'if-then' statements from the LP

A simple example provided below (see figure 3.1) illustrates how a topology of a boolean form of LP can be used to generate 'if-then' statements (rules) conveying knowledge in an explicit manner.

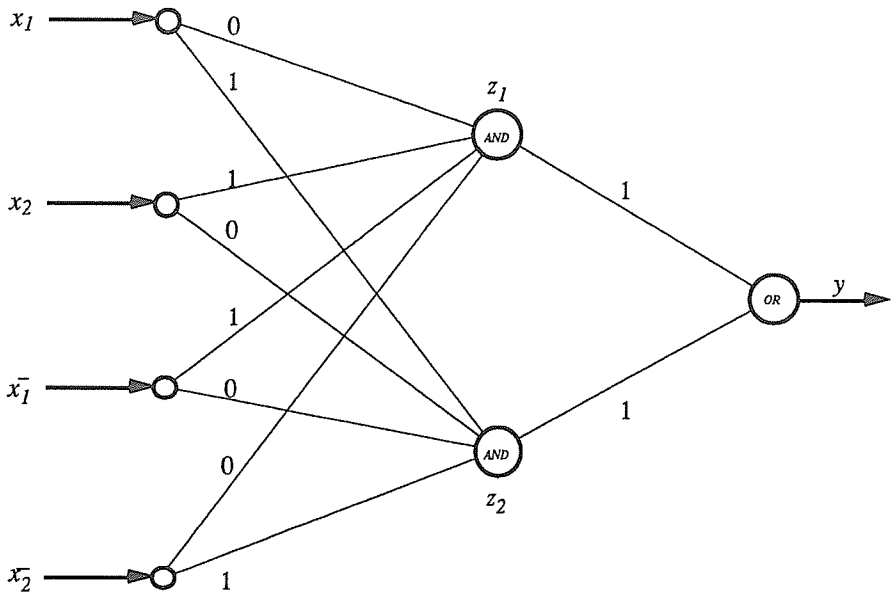


Figure 3.1 A 4-2-1 Logic Processor.

The output y of the LP can be described as

$$\begin{aligned} y &= (z_1 \; AND \; 1) \; OR \; (z_2 \; AND \; 1) \\ &= z_1 \; OR \; z_2 \end{aligned} \tag{3.20}$$

where

$$\begin{aligned} z_1 &= (x_1 \; OR \; 0) \; AND \; (x_2 \; OR \; 1) \; AND \; (\overline{x_1} \; OR \; 1) \; AND \; (\overline{x_2} \; OR \; 0) \\ &= x_1 \; AND \; \overline{x_2} \end{aligned} \tag{3.21}$$

$$\begin{aligned} z_2 &= (x_1 \; OR \; 1) \; AND \; (x_2 \; OR \; 0) \; AND \; (\overline{x_1} \; OR \; 0) \; AND \; (\overline{x_2} \; OR \; 1) \\ &= x_2 \; AND \; \overline{x_1} \end{aligned} \tag{3.22}$$

and therefore, the output y for the given LP in figure 3.1 becomes

$$y = \left(x_1 \; AND \; \overline{x_2}\right) OR \left(x_2 \; AND \; \overline{x_1}\right) \tag{3.23}$$

its equivalent rule-based notation reads as

if (x_1 and not x_2) or (x_2 and not x_1) then y.

that is, the output y is dependent on the present of input x_1 and the inhibitory of input x_2, or on the input x_2 and the inhibitory of input x_1. If we examine (3.23) carefully, we will realize that the given 4-2-1 (4 inputs, 2 hidden units, and 1 output) LP is nothing but a two input XOR function.

3.6 Software Simulations of LP

This section presents the works done by using software to simulate the logic processor. A program is written in C language and runs on SUN™ SPARC stations. Section 3.6.1 discusses a four input binary XOR problem A two input multivalued XOR problem is analyzed in section 3.6.2. We will be interested in showing the effects on using several triangular norms in the network implementation.

3.6.1 A Four Input Binary XOR Problem

A four input binary XOR problem as in the truth table form is shown below.

a	*b*	*c*	*d*	*Y*
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Table 3.5 Truth Table of a Four Input Binary XOR Function.

The output of the binary XOR is equal to 1 for an odd number of 1s at its inputs The values contained in Table 3.5 will be treated as a training set presented to a LP with eight input units, eight hidden units and one output unit (which denoted by 8-8-1 LP). The weights and the independent learning rate for each connection in the LP are initialized in the range of $[a(t), 1\text{-}a(t)]$ randomly. $a(t)$ is taken as described by (3.17).

Defining the product and probability sum as t and s-norms, with the use of off-line learning procedure and mean square error as the performance index, we found the LP has

achieved a mean square error less than 1.0e-7 at 5475 learning iterations. Perfect results were not obtained due to the fact that the weights in the LP are in the range of [*a(t)*, 1-*a(t)*], say[2.468e-6, 0.999]. However, using the boolean approximation technique listed in table 3.2, we can transform our network into a boolean version; that is, the entire weights in the induced boolean version network are either 0 or 1. We present the training set to this induced boolean version network again, and obtain the correct results as we expected, with the mean square error equal to 0. Figure 3.2 shows the graph of mean square error versus learning epochs for the LP. The same figure summarizes the MSE for four and six input-XOR problem.

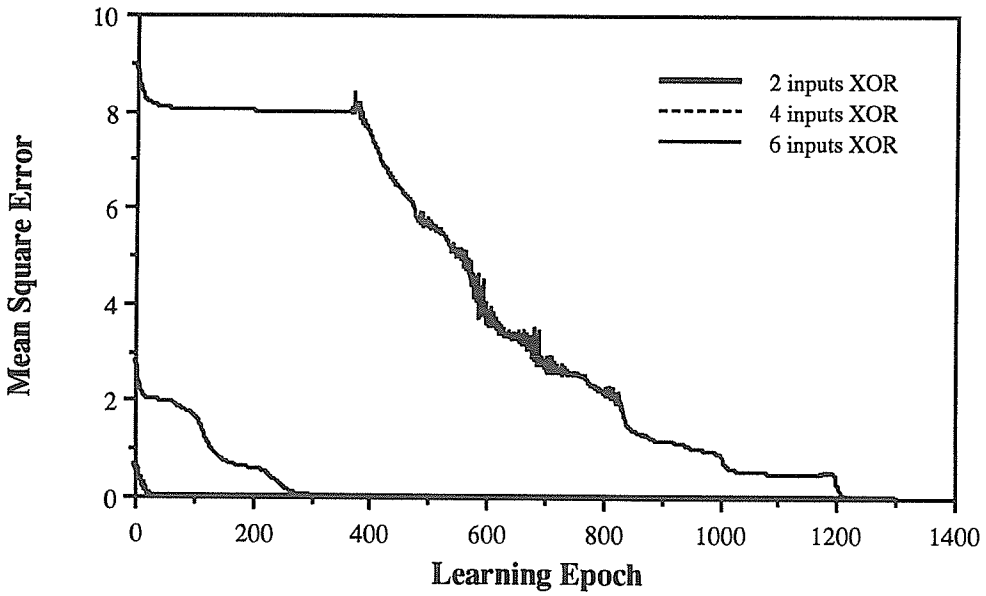


Figure 3.2 MSE in Successive Learning Epochs for Binary XOR Problem.

The rounded values of the weights obtained after 5475 learning epochs are:

		Hidden Layer							
		z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8
Input Layer	a	0.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0
	b	0.0	1.0	0.0	0.0	0.0	1.0	1.0	1.0
	c	0.0	0.0	1.0	0.0	1.0	1.0	1.0	0.0
	d	1.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0
	$\overline{a}$	1.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0
	$\overline{b}$	1.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0
	$\overline{c}$	1.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0
	$\overline{d}$	0.0	0.0	0.0	1.0	1.0	0.0	1.0	1.0

and the weights between the hidden and output layers are:

		Hidden Layer							
		z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8
Output Layer	y	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

The output y can be immediately derived in the form

$$y = \left[(z_1 \; AND \; 1.0) \; OR \; (z_2 \; AND \; 1.0) \; OR \; (z_3 \; AND \; 1.0) \; OR \; (z_4 \; AND \; 1.0) \; OR \; \ldots \right.$$
$$\left. \ldots \; OR \; (z_5 \; AND \; 1.0) \; OR \; (z_6 \; AND \; 1.0) \; OR \; (z_7 \; AND \; 1.0) \; OR \; (z_8 \; AND \; 1.0)\right]$$
$$= z_1 \; OR \; z_2 \; OR \; z_3 \; OR \; z_4 \; OR \; z_5 \; OR \; z_6 \; OR \; z_7 \; OR \; z_8$$

and the hidden element z_1 can be described by:

$$z_1 = \left[(a \; OR \; 0.0) \; AND \; (b \; OR \; 0.0) \; AND \; (c \; OR \; 0.0) \; AND \; (d \; OR \; 1.0) \; AND \; \ldots \right.$$
$$\left. \ldots \; AND \; (\overline{a} \; OR \; 1.0) \; AND \; (\overline{b} \; OR \; 1.0) \; AND \; (\overline{c} \; OR \; 1.0) \; AND \; (\overline{d} \; OR \; 0.0)\right]$$
$$= a \; AND \; b \; AND \; c \; AND \; \overline{d}$$

Similarly, the rest of the hidden elements are described by:

$$z_2 = c\ AND\ \overline{a}\ AND\ \overline{b}\ AND\ \overline{d}$$

$$z_3 = b\ AND\ \overline{a}\ AND\ \overline{c}\ AND\ \overline{d}$$

$$z_4 = b\ AND\ c\ AND\ d\ AND\ \overline{a}$$

$$z_5 = a\ AND\ b\ AND\ d\ AND\ \overline{c}$$

$$z_6 = a\ AND\ \overline{b}\ AND\ \overline{c}\ AND\ \overline{d}$$

$$z_7 = d\ AND\ \overline{a}\ AND\ \overline{b}\ AND\ \overline{c}$$

$$z_8 = a\ AND\ c\ AND\ d\ AND\ \overline{b}$$

By identifying z_i's with the corresponding entries of the Karnaugh map,

cd \ *ab*	00	01	11	10
00		z_3		z_6
01	z_7		z_5	
11		z_4		z_8
10	z_2		z_1	

Figure 3.3 Karnaugh Map for the Function *y*.

it becomes obvious from the map that the learned function is just a XOR expression. This example also shows the interpretation of the accumulated knowledge in the LP to a logical representation is easy to achieve. One further point we want to emphasis here is

that the LP had no prior domain knowledge or any information of the training set before the learning procedure proceed, and it can determine the logical relationships between the inputs and outputs data from the training set automatically.

3.6.2 A Two Inputs Multivalued XOR Problem

The second example that we will consider here is a multivalued XOR problem with the learning patterns summarized below:

x_1	x_2	Y
0.0	0.0	0.0
0.0	0.2	0.2
0.0	0.4	0.4
0.0	0.6	0.6
0.5	1.0	0.5
0.6	0.8	0.4
1.0	1.0	0.0
0.6	0.2	0.6
0.8	0.0	0.8

Table 3.6 Learning Set Used for a Multivalued XOR Problem.

We present this training set to a 4-2-1 LP. The weights of the network and the independent learning rates for each connection were initialized randomly as before. Defining the product and probability sum as t- and s-norms, the off-line learning procedure produced MSE = 0.002875 after 30,000 learning epochs. The rounded values

of the weights which are stored between the input and the hidden layers in the LP at 30,000 learning iterations are:

		Hidden Layer	
		z_1	z_2
Input Layer	x_1	0.991	0.015
	x_2	0.000	1.000
	$\overline{x_1}$	0.000	1.000
	$\overline{x_2}$	1.000	0.007

and the weights between the hidden and output layers are:

		Hidden Layer	
		z_1	z_2
Output Layer	y	1.000	1.000

We apply the boolean approximation technique to the LP; the induced boolean version of the LP is obtained at $\alpha = 0.9$ and $\beta = 0.9$. The weights stored in the boolean version LP become:

		Hidden Layer	
		z_1	z_2
Input Layer	x_1	1	0
	x_2	0	1
	$\overline{x_1}$	0	1
	$\overline{x_2}$	1	0

		Hidden Layer	
		z_1	z_2
Output Layer	y	1	1

while the mean square error of this boolean version LP with the training set is now equal to 0.003075. The logical relationship inferred from the network reads as

$$y = \left(x_1 \; AND \; \overline{x_2}\right) OR \left(\overline{x_1} \; AND \; x_2\right)$$

When we looked at the learning set again, we found that most of the training patterns are non-interactive. Therefore, the use of *Min* and *Max* as the t- and s-norms operators seems to be legitimate. By doing that, we can expect a lower performance index and much faster learning. In order to verify that, we re-run the original LP with the same initial conditions, except the t- and s-norms that have been taken as *Min* and *Max* respectively. We found that the new LP is characterized by the mean square error less than 1.0e-7 after 1000 learning epochs, and the weights are:

		Hidden Layer	
		z_1	z_2
Input Layer	x_1	0.629	0.000
	x_2	0.000	0.953
	$\overline{x_1}$	0.000	0.800
	$\overline{x_2}$	0.600	0.000

		Hidden Layer	
		z_1	z_2
Output Layer	y	0.819	0.793

The values of MSE in the successive learning epochs for both versions of the LP are shown in figure 3.4.

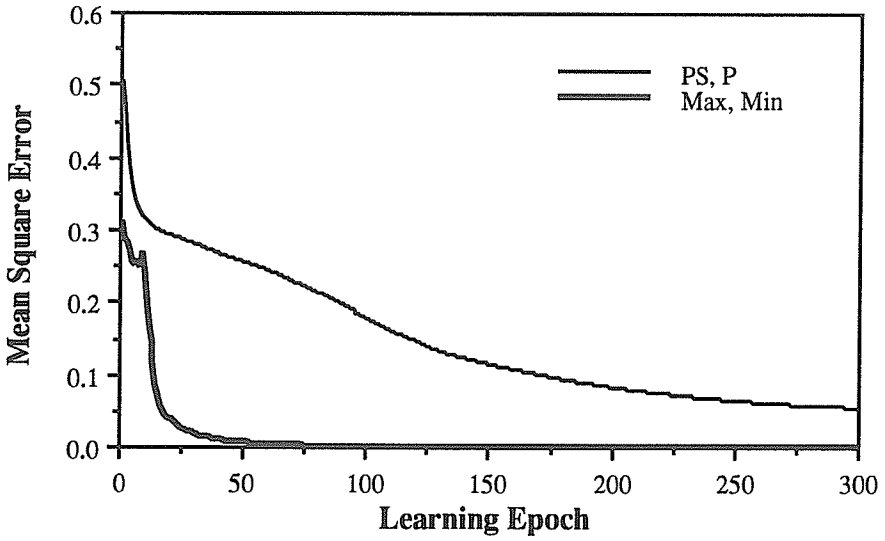


Figure 3.4 MSE in Successive Learning Epochs for A Two Input Multivalued XOR Problem.

The boolean approximation of the *Max, Min* version of the LP, yields $\alpha = 0.79$, $\beta = 0.59$. Its weights are specified below:

Hidden Layer

Input Layer	z_1	z_2
x_1	1	0
x_2	0	1
$\overline{x_1}$	0	1
$\overline{x_2}$	1	0

Hidden Layer

Output Layer	z_1	z_2
y	1	1

The boolean *Max, Min* version of LP has a performance index (using MSE) equal to 0 with the training set. Compared to the performance index 0.003075 produced by the boolean *PS, P* version of LP, we found that there is a significant improvement of the output. Again, the boolean *Max, Min* version of the LP can be described concisely as

$$y = \left(x_1 \; AND \; \overline{x_2}\right) OR \left(\overline{x_1} \; AND \; x_2\right)$$

In conclusion, we found both versions (*PS, P* version and the *Max, Min* version) of the LP produced the same logical expression from the training patterns. However, the *Min, Max* version LP shows a much better performance than the other one. This is simply because our learning patterns are non-interactive, which are more suitable to use *Min* and *Max* operators to implement the network. This also suggests that the selection of t- and s-norms should be application oriented which complies with our comments made in section 3.3.1.

3.7 Conclusions

We have introduced the Logic Processor (LP) and studied its basic architecture. The learning algorithm along with the two distinct performance indices were also proposed. Translation of weights stored in the LP to a logical knowledge representation is also illustrated. Software simulations for the LP with two examples were also shown. The use of logic processor will be discussed in the next chapter.

Chapter 4
Distributed Modeling

4.1 Distributed Modeling

In system modeling and identification process, we are usually provided with a vast amount of experimental data. The aim of identification is to describe a model that can govern the physical phenomenon. The traditional modeling technique is used to derive a mathematical model to describe the system behavior based on these experimental data. Such a mathematical model is constructed by analytical linear or nonlinear functions, and the behavior of the system is described by the exact values of its variables (like forces, positions, etc.) at each time instant. Although in most of the cases, the mathematical model can describe the system completely, it fails to provide much insight into how the system works. The insightful concepts and distinctions are usually qualitative, but they are embedded with the much more complex framework established by the real valued variables and differential equations [3].

From the discussion of the LP in the last chapter, we learned that the strong logical structure in the LP can formulate the logical relationship underlying the given data set. This provides us with an alternative way to describe a system. Without involving any complicated mathematical equations, we can use the logical relationships among the

state variables and inputs established by the LP to describe a system. This inspires the idea of distributed modeling. The aim of distributed modeling is to depart from a single analytical and fully centralized model and build a family of loosely coupled local models to describe the performance of the state variables of the considered system [14]. A separate LP is assigned to each of the state variables in the considered system. Its task is to describe the logical interactions of the modeling state variable with the other state variables and inputs to the system. Using the LP as building blocks in distributed modeling, we can derive a model in a far more simpler, but nevertheless formal, qualitative fashion. These new models are easy to understand and they provide further information for reasoning and analyzing the system qualitatively.

4.2 Topology of Distributed Modeling

Let us consider a system with m state variables, $x_1, x_2, ..., x_m$; and n inputs $i_1, i_2, ..., i_n$ taking on values in [0, 1],

$$\forall_i x_i \in [0,1] \qquad \& \qquad \forall_j i_j \in [0,1] \tag{4.1}$$

The distributed model

$$\mathbb{D} = (\mathrm{LP}_1, \mathrm{LP}_2, \ldots, \mathrm{LP}_m) \tag{4.2}$$

is defined as a family of fully or partially connected logic processors, LP_1, LP_2, ..., LP_m,. See figure 4.1.

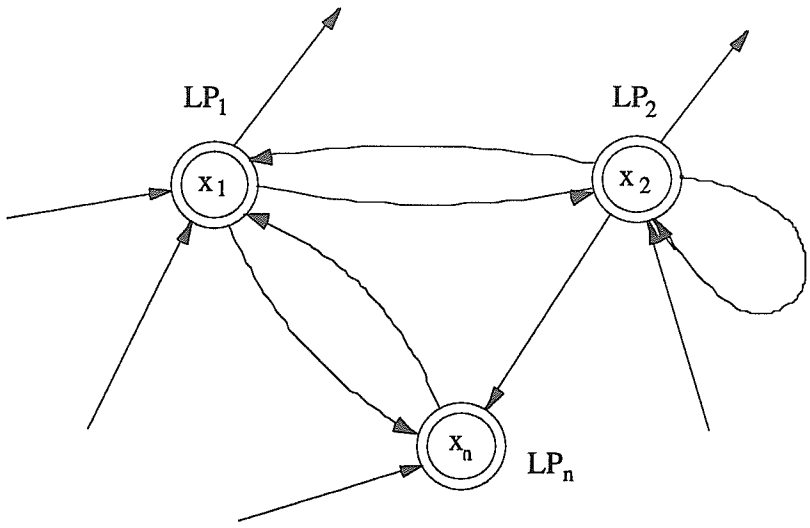


Figure 4.1 A Distributed Model.

The input layer of the LP_i consists of all the inputs $i_1, ..., i_n$, state variables $x_1, ..., x_m$, as well as their complements $\bar{i}_1, \ldots, \bar{i}_n$ and $\bar{x}_1, \ldots, \bar{x}_m$. The hidden layer consists of h hidden elements, and the output layer comprises of a single node representing the state variable x_i. More precisely, this i-th logic processor models the dynamical behavior of the variable x_i by describing its evolution in discrete time instants as

$$x_i(t+1) = \mathrm{LP}_i\left(x(t), i(t), \bar{x}(t), \bar{i}(t)\right) \tag{4.3}$$

Figure 4.2 shows a typical LP configuration that models the state variable x_i.

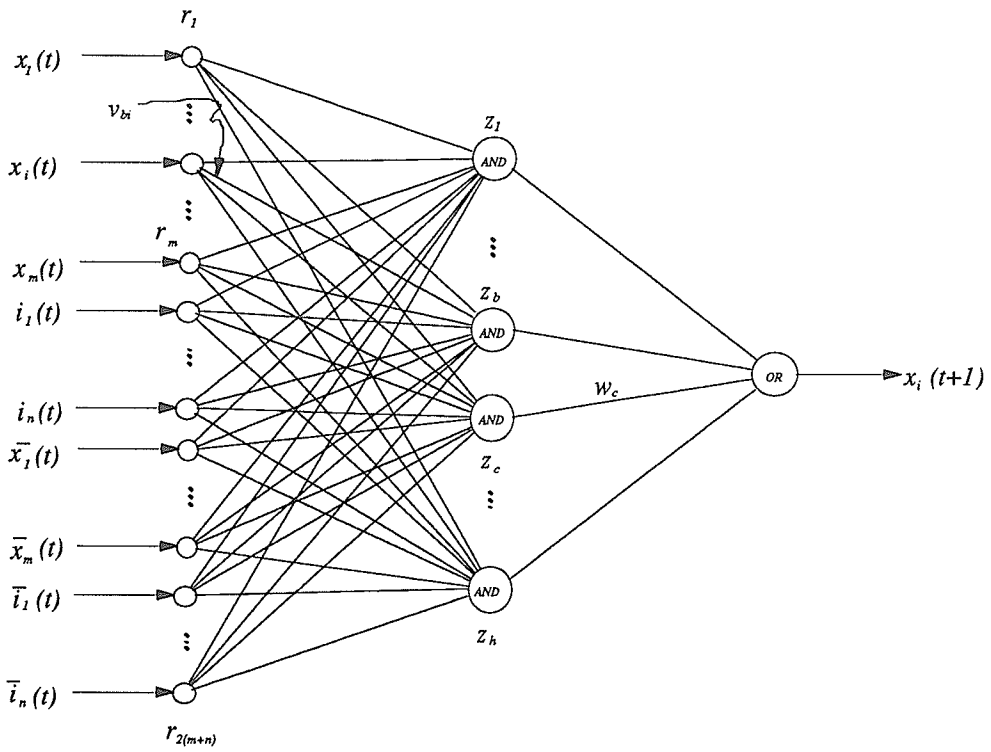


Figure 4.2 A Typical Configuration of LP_i used in Distributed Modeling.

The output $x_i(t+1)$ of the LP_i can be expressed by

$$x_i(t+1) = \sum_{j=1}^{h} \left(w_j(t)\ \ t\ \ z_j(t) \right) \tag{4.4}$$

where w_j is the strength of the connection between element j in the hidden element to the output element, z_j is the output from the element j in the hidden element, and is obtained by

$$z_j = \prod_{b=1}^{2\cdot(m+n)} \left(v_{jb}(t) \, s \, x_b(t) \right) \tag{4.5}$$

with v_{jb} denoting the connection between the node b in the input layer to the hidden element j in the hidden layer; $x_i(t)$ represents the i-th input in the input layer at time t.

In some cases, we may want to include a higher order dynamics in modeling the system. This can be done in the LPs by introducing extra inputs in the input layer. For a LP which models p-th order dynamics of the state variable x_i, its evolution in discrete time instants can be described as

$$\begin{aligned} x_i(t+1) = \mathrm{LP}_i\big[& x(t), i(t), x(t-1), x(t-2), \ldots, x(t-p+1), \\ & \bar{x}(t), \bar{i}(t), \bar{x}(t-1), \bar{x}(t-2), \ldots, \bar{x}(t-p+1) \big] \qquad p = 1, 2, \ldots \end{aligned} \tag{4.6}$$

Furthermore for a realistic system, the changes of inputs may require some time to affect the state variables. This period of time is known as delay, and will be denoted by τ. The first order LP can cope with such behavior of the system by presenting the delayed inputs to the LP,

$$x_i(t+1) = \mathrm{LP}_i\big[x(t), i(t-\tau), \bar{x}(t), \bar{i}(t-\tau) \big] \qquad \tau = 1, 2, \ldots \tag{4.7}$$

4.3 A Gas Furnace Example

In this section, we will employ a realistic example to demonstrate the uses of distributed modeling. Consider a gas furnace system [1] in which air and methane are

combined to form a mixture of gases containing CO_2 (carbon dioxide) as shown in figure 4.3.

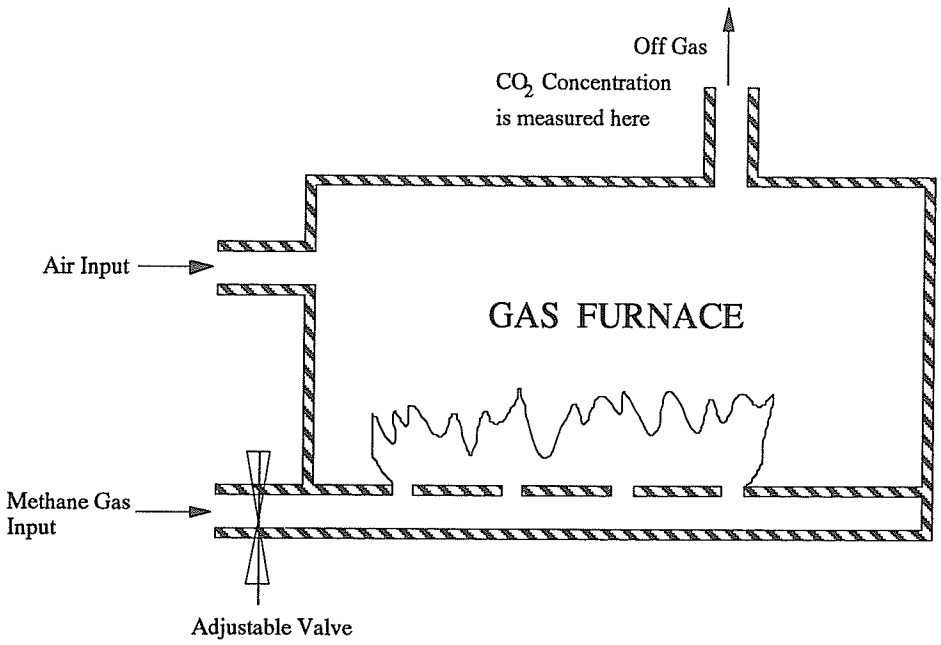


Figure 4.3 A Gas Furnace System.

Air fed to this gas furnace is kept constant, while the methane feed rate can be varied in any desired manner and the resulting CO_2 concentration in the off gases is measured. A time series used for identification purposes consists of 296 successive pairs of observations: the methane gas feed rate and the CO_2 concentration in the off gases are recorded and shown in Appendix C.

Before proceeding to the construction of logic processor, the experimental data should be coded into a [0, 1] format as required in this modeling. We will be concerned with a nonlinear normalization through the use of linguistic variables.

Linguistic Variable

Briefly speaking, linguistic variables [2][28] are variables that take on values from a finite family of fuzzy sets (linguistic labels). These linguistic labels are often referred to as generic terms. For example, the term set {*low, medium, high*} contains the terms that can be assigned to the linguistic variable CO_2 *concentration*. Each term in the term set is a fuzzy set, and is generated by a syntactic rule associated with the variable. Considering the linguistic variable CO_2 *concentration* again, let the underlying numeric variable (universe of discourse) is distributed between 10% and 90%. We might choose *low, medium* and *high* as our terms in the term set. Taking {10, 20, 30, 40, 50, 60, 70, 80, 90} as the discrete universe **X**, the terms might be defined as

$$low = \left\{ 1/10 + 0.8/20 + 0.6/30 + 0.4/40 + 0.2/50 \right\}$$

$$midium = \left\{ 0.2/30 + 1/40 + 1/50 + 1/60 + 0.2/70 \right\}$$

$$high = \left\{ 0.2/50 + 0.4/60 + 0.6/70 + 0.8/80 + 1.0/90 \right\}$$

With the aid of fuzzy conditional statements, we can generate linguistic descriptions of some simple relationships between linguistic variables. For example,

if the *methane feed rate* is *positive*, and the CO_2 *concentration* is *low* then the CO_2 *concentration* at next time instant will *increase*.

Fuzzy conditional statements describe the relationships between linguistic variables. In this way, both linguistic variables and fuzzy conditional statements provide a formal framework of knowledge representation necessary to reason qualitatively about the system.

Simulation Procedure

Now, let us return to our gas furnace data. We use the linguistic variable *methane feed rate* to describe the input *x(t)*, with the generic terms *x_negative*, *x_zero* and *x_positive* as elements in the term set. The membership functions of the terms are shown in figure 4.4.

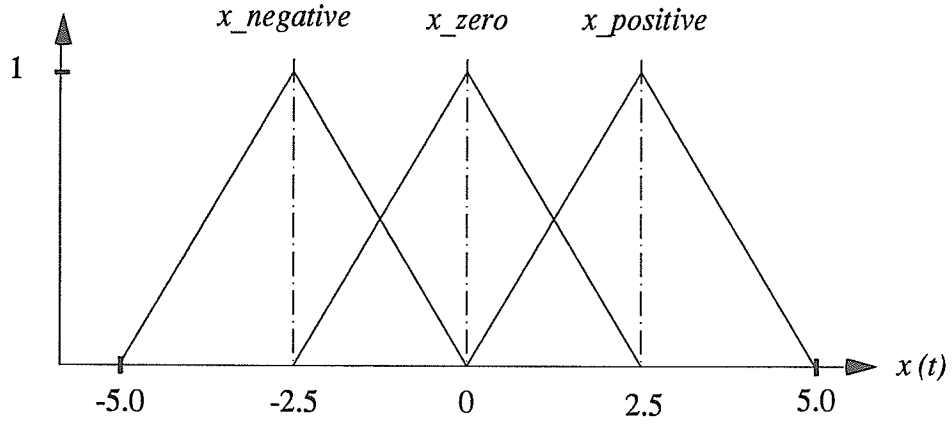


Figure 4.4 Membership Functions of Generic Terms of Methane Feed Rate in Gas Furnace System.

We use the linguistic variable *CO_2 concentration* to describe the state variable *y(t)* with the terms *y_low*, *y_medium* and *y_high* in the term set, see figure 4.5.

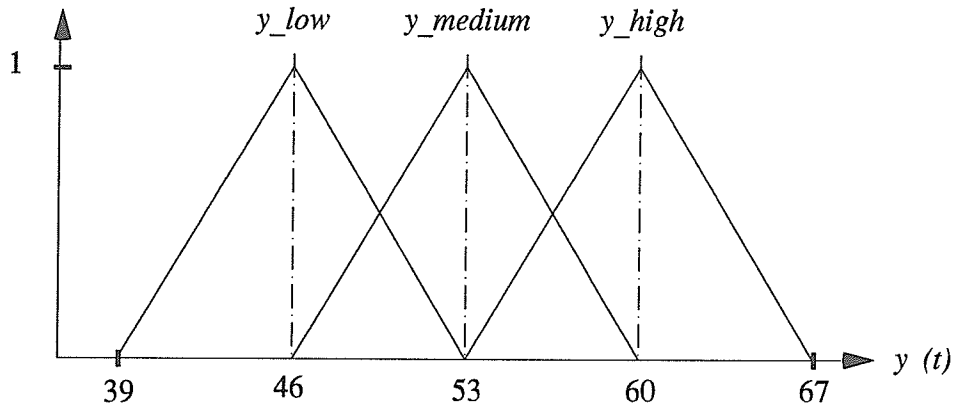


Figure 4.5 Membership Functions of Generic Terms of CO_2 Concentration in Gas Furnace System.

Each individual data in the input *x(t)* is represented by the fuzzy sets *x_negative*, *x_zero* and *x_positive* based on the membership functions given above. For example, for *x(t)* = 0, this numerical value is translated into membership values of the terms as follows:

$$\mu_{x\_negative}(0) = 0$$

$$\mu_{x\_zero}(0) = 1$$

$$\mu_{x\_positive}(0) = 0$$

We perform a similar transformation task for the state variable $y(t)$ as well. As a result, there are three fuzzy sets for inputs, and three fuzzy sets for the state variables in our gas furnace system.

To build a distributed model for the gas furnace system in this setting, we need to use three logic processors. Each LP is associated with a term (fuzzy set) defined for the linguistic variable *CO_2 concentration*. For simplicity, we will use the LPs to model the second order of dynamics of the state variables in the system. The LPs used in the gas furnace distributed model have eighteen input units, eight hidden units and one output unit (a 18-8-1 LP). Figure 4.6 shows a block diagram of the logic processor LP_{y_low} that models the fuzzy set *y_low*.

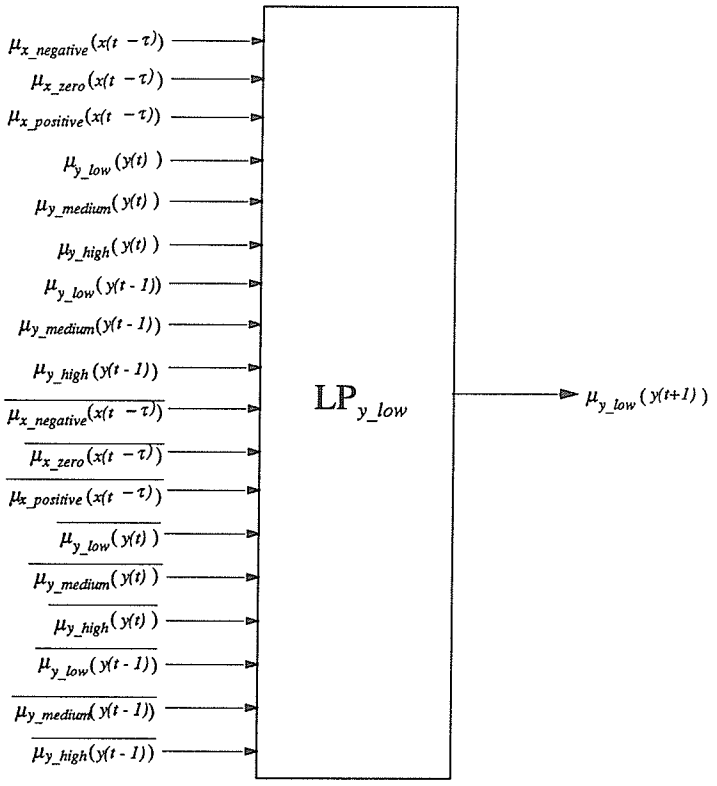


Figure 4.6 Block Diagram for LP_{y_low}.

We divide the whole identification process into two different phases: learning and testing. In the learning phase, we train the LPs with the first 250 data patterns. The testing phase involves the entire data set including those 35 data patterns that have not been utilized during the training phase.

Prior to the learning process, we have no information on the delay time τ that makes the *CO2 concentration* take effects on changing the *methane feed rate*. Therefore,

we need to run several simulations with different delay time to determine the best model of the system. The weights and the independent learning rates in the LPs for all the simulations are initialized randomly to small numbers in [0, 1]. The product and probabilistic sum are specified as the instances of triangular norms. After 10,000 learning epochs, the values of performance index (MSE) achieved for different delay times are summarized below:

τ	LPy_low	LPy_medium	LPy_high
0	0.256367	0.686035	0.659286
1	0.154256	0.454829	0.580235
2	0.080373	0.286883	0.539959
3	0.118306	0.314820	0.564423
4	0.340463	0.661920	0.705799
5	0.474118	0.930187	0.796542

As we observed from the above table, the LPs with $\tau = 2$ produced the lowest MSE. This value $\tau = 2$ will be used in the LPs to model the second order dynamics. The weights of the these LPs are provided in Appendix D.

In order to facilitate interpretation of the models and reveal the most significant relationships captured by the networks, we apply the optimal threshold level technique described in section 3.4.1. The threshold levels leading to its optimal boolean version of LPs are now equal to:

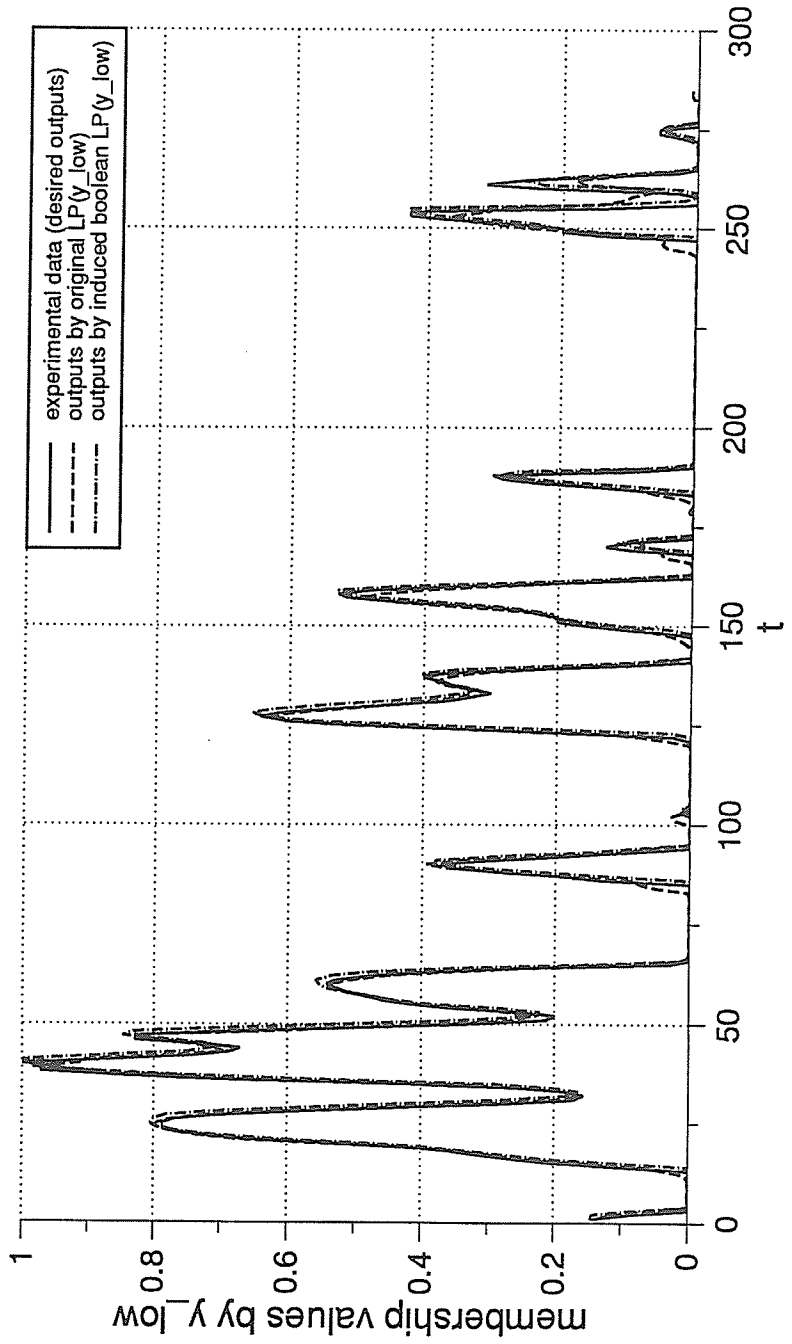
	LP_{y_low}	LP_{y_medium}	LP_{y_high}
Threshold Level	$\alpha = 0.9$ $\beta = 0.7$	$\alpha = 0.9$ $\beta = 0.6$	$\alpha = 0.9$ $\beta = 0.7$

The corresponding weights of the induced boolean LPs are included in Appendix D.

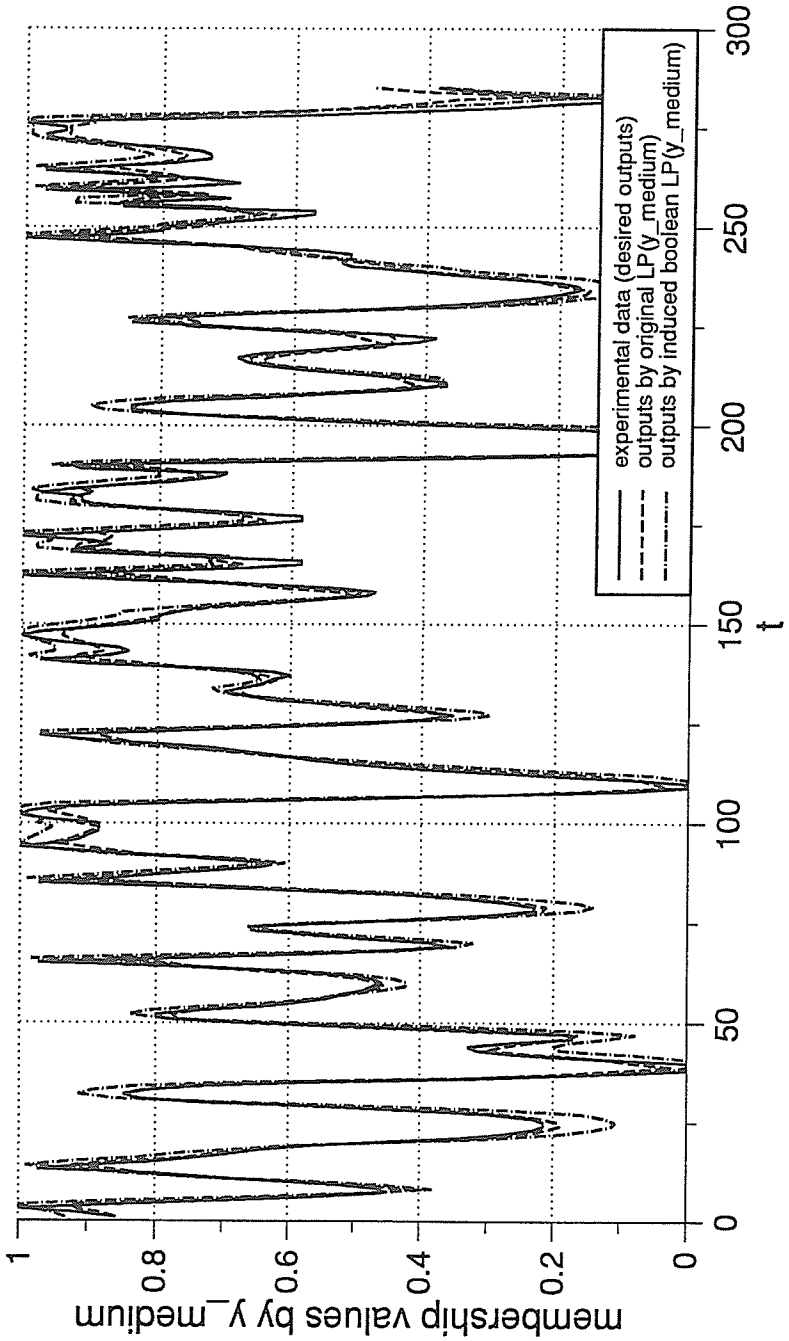
Obviously, the induced boolean versions constitute approximations of the LPs and could lead to higher MSE values. This is in fact illustrated in the table below using the testing data set:

	LP_{y_low}	LP_{y_medium}	LP_{y_high}
P.I. of Original LPs	0.153529	0.501393	0.757056
P.I. of Boolean LPs	0.426270	1.036869	0.902215

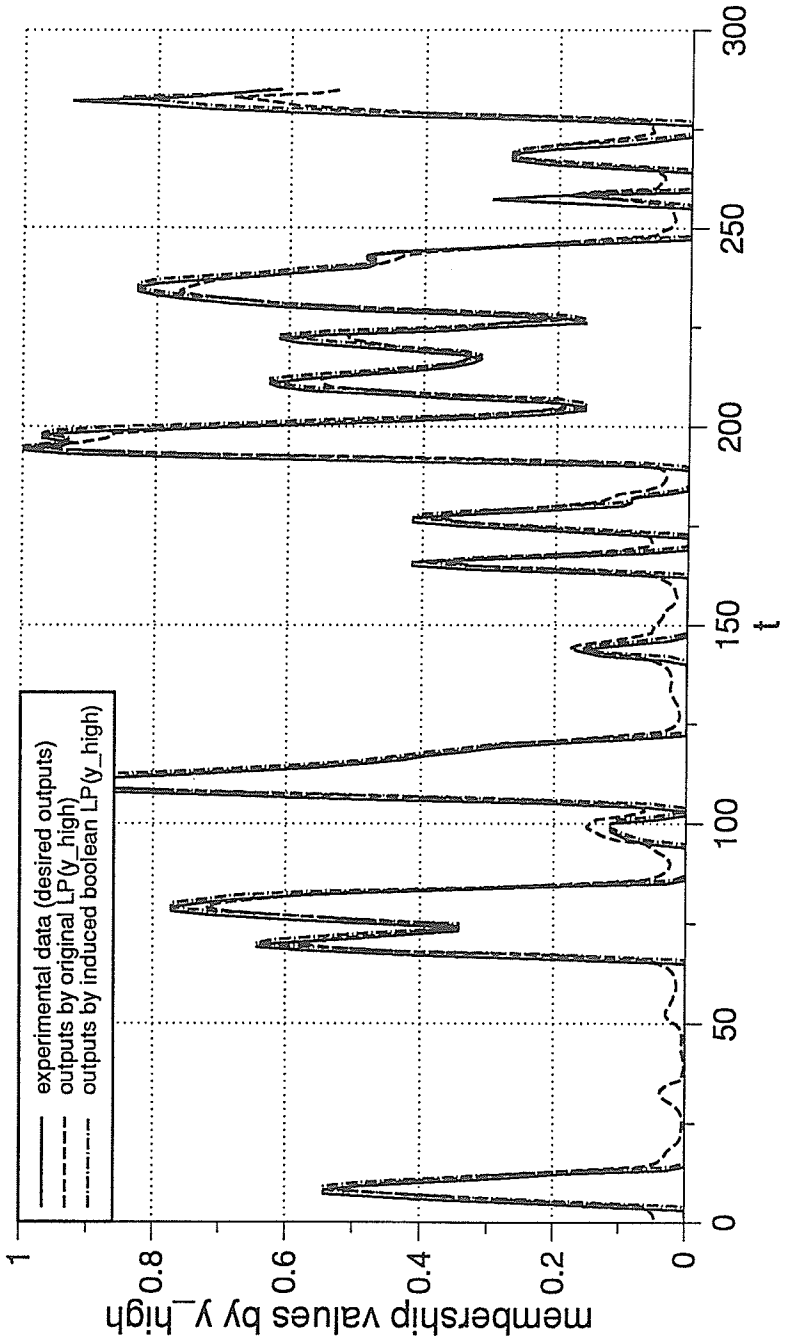
The following three figures illustrate the outputs produced by the original LPs and their induced boolean versions of LPs. Figure 4.7, 4.8 and 4.9 display the outputs of the LP_{y_low}, LP_{y_medium} and LP_{y_high}, respectively. Since the outputs of the LPs are the values of activation of the state linguistic variable (namely *y_low*, *y_medium*, and *y_high*), in order to come up with numerical outcomes of distributed modeling, we require another procedure that acts in a complementary way to that applied before. Its role is to convert three grades of activation into a single numerical value; the center of gravity is one of the methods being in a common use [16]. It results from averaging the model values of *y_low*, *y_medium* and *y_high* (denoted by r_1, r_2 and r_3 respectively). That is, based on



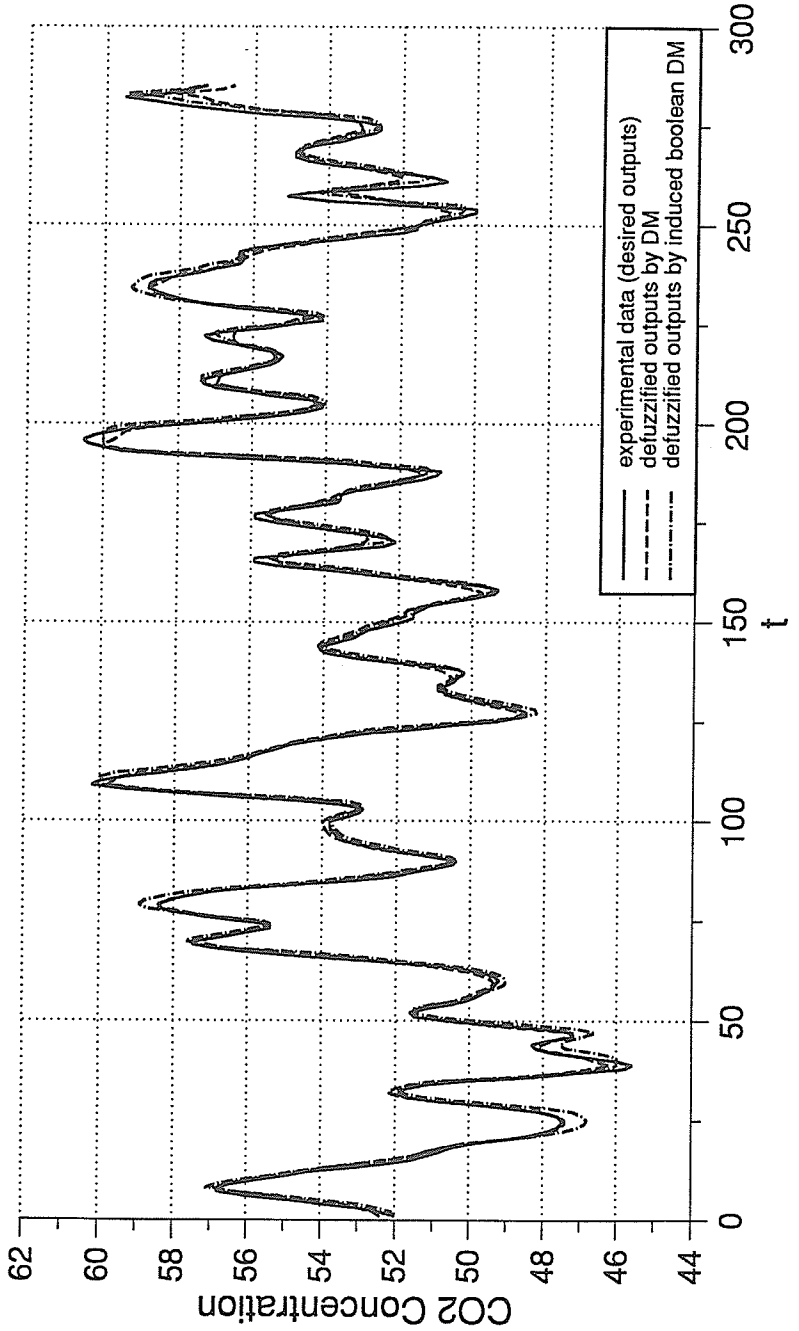
4.7 Membership Values of *y_low* in Successive Discrete Time Moments.



4.8 Membership Values of *y_medium* in Successive Discrete Time Moments.



4.9 Membership Values of *y_high* in Successive Discrete Time Moments.



4.10 Comparison of Defuzzified Outputs from the Distributed Model and the Desired Outputs.

their activation levels resulting from the LPs, say y_1, y_2 and y_3. The converted numerical value of the distributed model is calculated by

$$\tilde{y} = \frac{y_1 \cdot r_1 + y_2 \cdot r_2 + y_3 \cdot r_3}{y_1 + y_2 + y_3} \qquad \tilde{y} \in \mathbb{R} \tag{4.8}$$

Figure 4.10 shows the converted data from the distributed model and the original experimental data.

4.3.1 Rule-based System Description

We can interpret the interactions among the inputs and the state variables generating a series of logical expressions based on the connections in the LPs. By a straight forward inspection of the structure of the boolean approximations of the LPs, we derive,

$$\begin{aligned} y\_low(t+1) = {} & x\_positive(t-2) \cdot \overline{x\_zero}(t-2) \cdot y\_low(t-1) \cdot \overline{y\_low}(t-1) \cdot \overline{y\_high}(t-1) + \\ & \overline{x\_negative}(t-2) \cdot y\_low(t) \cdot \overline{y\_high}(t). \end{aligned} \tag{4.9}$$

$$\begin{aligned} y\_medium(t+1) = {} & x\_zero(t-2) \cdot x\_positive(t-2) \cdot \overline{x\_zero}(t-2) \cdot y\_medium(t) \cdot \overline{y\_medium}(t) \cdot \\ & \cdot y\_low(t-1) \cdot y\_medium(t-1) \cdot \overline{y\_low}(t-1) \cdot \overline{y\_medium}(t-1) + \\ & \overline{x\_positive}(t-2) \cdot y\_medium(t) \cdot \overline{y\_high}(t) + \\ & \overline{x\_negative}(t-2) \cdot y\_medium(t) \cdot \overline{y\_low}(t). \end{aligned} \tag{4.10}$$

$$y\_high(t+1) = \overline{x\_positive}(t-2) \cdot y\_high(t) \cdot \overline{y\_low}(t) \cdot \overline{y\_low}(t+1). \tag{4.11}$$

Some of the terms standing in the above expressions can be simplified. For example, the linguistic term in (4.11), $y\_high(t) \cdot \overline{y\_low}(t)$ can be simplified to $y\_high(t)$

considering their membership functions that result from these expressions, see also figure 4.11.

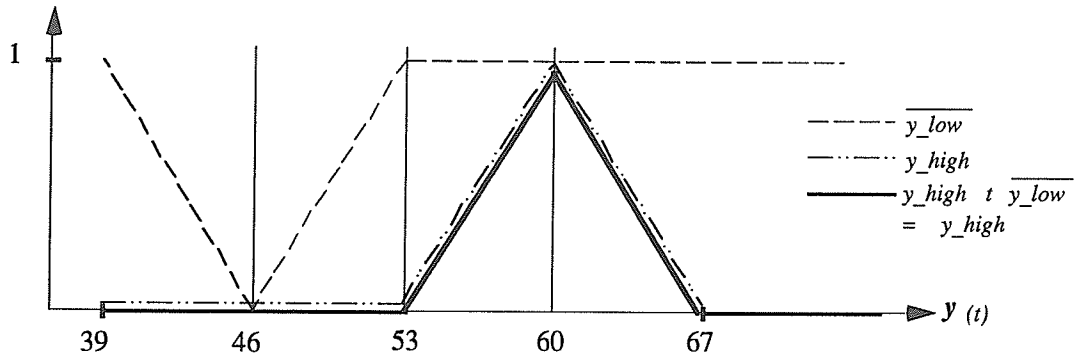


Figure 4.11 Simplification of the Membership Functions.

The results after this simplification are given below:

$$y\_low(t+1) = \overline{x\_negative}(t-2) \cdot y\_low(t). \tag{4.12}$$

$$y\_medium(t+1) = \overline{x\_positive}(t-2) \cdot y\_medium(t) \cdot \overline{y\_high}(t) + \overline{x\_negative}(t-2) \cdot y\_medium(t) \cdot \overline{y\_low}(t). \tag{4.13}$$

$$y\_high(t+1) = \overline{x\_positive}(t-2) \cdot y\_high(t) \cdot \overline{y\_low}(t+1). \tag{4.14}$$

The family of conditional statements involving the linguistic terms that are equivalent to (4.12) to (4.14) are illustrated in Table 4.1.

if *methane feed rate* at time = t-2 is $\overline{x\_negative}$ and CO_2 *concentration* at time = t is *y_low* then CO_2 *concentration* at time = $t+1$ is *y_low*.
if *methane feed rate* at time = t-2 is $\overline{x\_positive}$ and CO_2 *concentration* at time = t is *y_medium* and $\overline{y\_high}$ then CO_2 *concentration* at time = $t+1$ is *y_medium*.
if *methane feed rate* at time = t-2 is $\overline{x\_negative}$ and CO_2 *concentration* at time = t is *y_medium* and $\overline{y\_low}$ then CO_2 *concentration* at time = $t+1$ is *y_medium*.
if *methane feed rate* at time = t-2 is $\overline{x\_positive}$ and CO_2 *concentration* at time = t is *y_high* and CO_2 *concentration* at time = t-1 is $\overline{y\_low}$ then CO_2 *concentration* at time = $t+1$ is *y_high*.

Table 4.1 Conditional Statements for the Gas Furnace System.

The conditional statements in Table 4.1 give us an insight of how this gas furnace system works. As we go over these statements, we can conclude that the *methane feed rate* has a negative effect on the CO_2 *concentration* with delay $\tau = 2$. This makes perfect sense since air is fed at a constant rate in the considered system. Therefore if the *methane feed rate* is too high, the chemical reaction inside the gas furnace will not be completed and some of the methane is left un-burned. This un-burned methane will leave the gas furnace as a off gas and thus decreasing the concentration of the CO_2. On the other hand, if the *methane feed rate* is low, the gas furnace can have the methane burned completely before it leaves the gas furnace, and so it increases the concentration of the CO_2. Also, the chemical reaction in the gas furnace takes time to complete, and thus lead to delay time $\tau = 2$.

4.3.2 Distributed Modeling and Regression Models

The Box-Jenkins data constitutes a widely used benchmark for testing identification methods, in particular for time series analysis. An example of a regression model obtained in [1] is given in the form:

$$\begin{aligned}\hat{y}_t(l) = {} & 2.1[y(t+l-1)] - 1.5021[y(t+l-2)] + 0.3591[y(t+l-3)] \\ & -0.53[x(t+l-3)] + 0.4409[x(t+l-4)] - 0.2778[x(t+l-5)] \\ & +0.5472[x(t+l-6)] - 0.3213[x(t+l-7)] \\ & +[a(t+l)] - 0.57[a(t+l-1)]\end{aligned} \tag{4.15}$$

where l is the lead forecast and $\hat{y}_t(l)$ denotes the lead forecast value for $y(t)$ based at time t. $[a(t+l)]$ stands for the noise correction term at time $t+l$, and is calculated as:

$$[a(t+l)] = \begin{cases} y(t+l) - \hat{y}_t(l) & l \le 0 \\ 0 & l > 0 \end{cases} \tag{4.16}$$

This regression model is used to compare the results obtained from the gas furnace distributed model. The outputs by the induced boolean LPs are converted to numerical values through (4.8). Figure 4.9 shows the outputs by the regression model and the ones produced by the boolean distributed model that are generated the data patterns 250 to 285 in the data set. The desired outputs for the gas furnace system are also included for comparison.

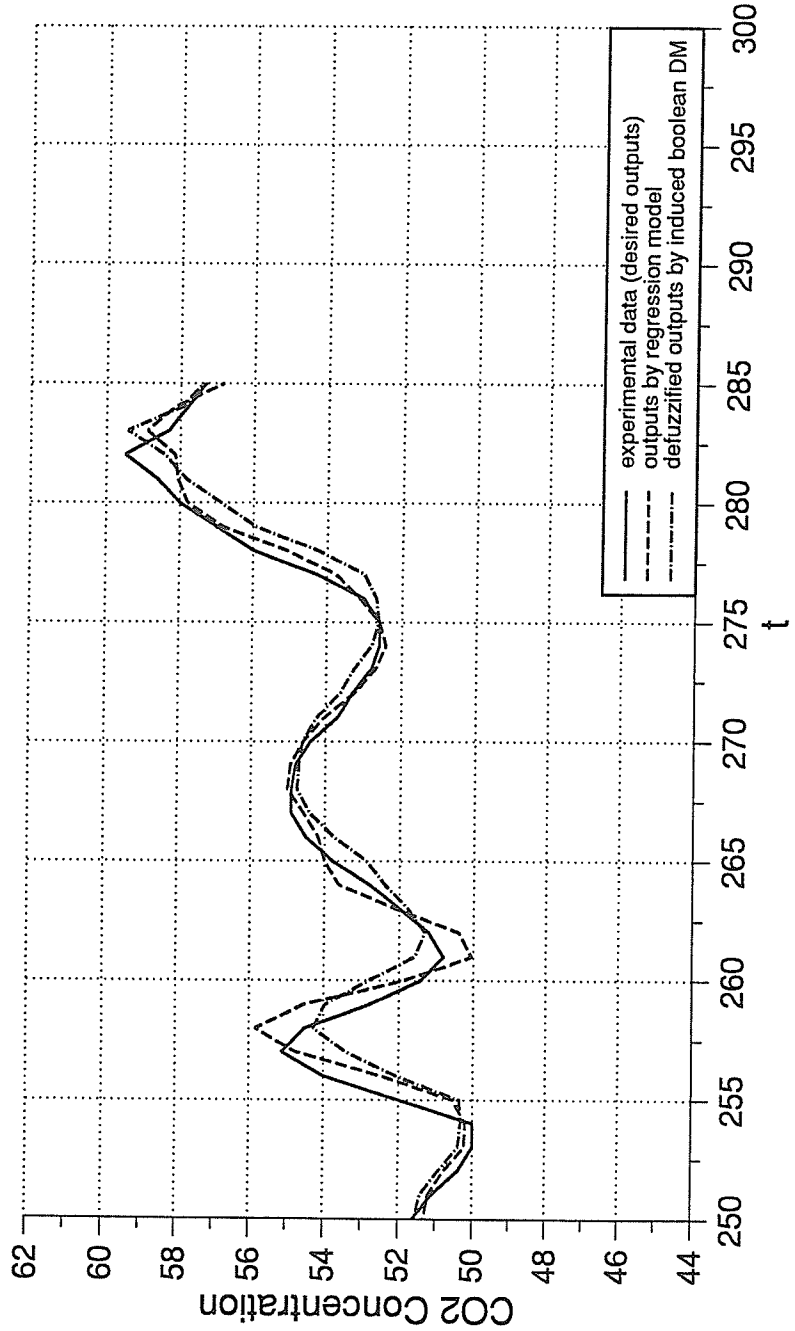
The performance indices (using MSE) for the induced boolean distributed model and the regression model are listed below.

	Regression Model	Distributed Model
MSE	8.243204	56.413353

Both the results shown in figure 4.11 tend to follow the desired data patterns. However, the regression model requires a relatively long period of the previous data to be included to generate any prediction (this time horizon is three and seven data points for the input and state variable, respectively). In contrast to the distributed model, the expression in (4.15) does not give any insight into how the gas furnace works, but the descriptions in table 4.1 given by the distributed model show how the inputs and state variables interact qualitatively.

4.3.3 Simulation Summary

A gas furnace system is used to illustrate the distributed modeling. The values of the input and state variables are transformed into an appropriate form (linguistic terms) and afterwards handled by the LPs. A distributed model is built to learn the interactions among the linguistic terms of the inputs and the state variables. The encoded knowledge in the model is then interpreted as a collection of fuzzy conditional statements. In summary, the relationships between the input and state variables of the gas furnace system are interpreted successfully from the distributed model, and the overall performance of this model is good. The MSE value could be made lower by increasing the number of linguistic terms defined for the variables.



4.12 Results of Modeling Produced by the Induced Boolean Distributed Model and by the Regression Model.

4.4 Conclusions

We have introduced the concepts of distributed modeling in this chapter. The topology and the construction of distributed modeling were discussed. The distributed model of a gas furnace system has been built. The results are good both from a perspective of purely numerical point of view as well as a qualitative behavior of the model.

Chapter 5
Conclusions and Recommendations

5.1 Final Conclusions

In this thesis, the basic issues of logic processor and the concepts of distributed modeling are presented. The breeding of the concepts and techniques of neural networks and fuzzy sets provides an interesting way of data processing. We have studied a new class of fuzzy neural networks called logic processors, which are constructed with the use of a set of logic-oriented neurons. This logic-based network is capable of capturing and encoding logical relationships conveyed by the problem at hand. The LP is purely logic driven, and its structure after learning can be interpreted explicitly, yielding a collection of 'if-then' statements.

Distributed modeling provides an alternative way to look at a system. The obtained distributed model of a system uses logical expressions to describe its behavior, rather than a collection of analytical linear or nonlinear functions. The logical expressions provided by the distributed model can be interpreted as the collections of "if-then" statements, which gives a far more simpler, but nevertheless formal way of system description. While the conceptual and algorithms aspects of the LP and distributed modeling have been thoroughly studied, the problem of applicability of the LP in

different domains should be addressed in the future. In conclusion, the objectives of the thesis have been achieved.

5.2 Recommendations

The POM version of the LP is correlated with its dual SOM version. It is worthwhile to investigate these links and verify if one can improve the model description by combining the information obtained from these two networks. Also, the architecture of the LP is flexible. The *AND-OR* fuzzy neural network studied here is only one among many possible configurations. A vast number of different classes of fuzzy neural networks can be constructed from the proposed set of logic-oriented neurons to cope with various applications [18]. The inherent logical background of the problem at hand can be directly mapped into the logic-oriented network by choosing appropriate logic-based neurons and arrange them into a proper architecture. This structure of the network can provide further extension along the field.

Finally, an area that has not been examined in detail in the thesis is to derive a formal and complete framework to work with distributed modeling. Such a general framework may be perceived as composed of three conceptual units: the input interface, the processing block and the output interface. The role of the input interface is to transform the input information into an appropriate format to be handled by the processing block. The objective of the processing block is to build a distributed model to describe the system. The output interface involves the interpretation of the results from

the processing block to a form of easily comprehended explicit logical constructs, like 'if-then' statements.

References

[1] Box, G. E. and G. M. Jenkins. *Time Series Analysis: Forecasting and Control*, Holden-Day, California, 1970, pp. 370-408.

[2] D'Ambrosio, B. *Qualitative Process Theory Using Linguistic Variables*, Springer-Verlag, New York, 1989, pp. 56-58.

[3] De Kleer, J. and J. Brown. *A Qualitative Physics Based on Confluences*, Artificial Intelligence, Volume 24, Number 1-3, December 1984, pp. 7-83.

[4] Folger, T. and G. J. Klir. *Fuzzy Sets, Uncertainty and Information*, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[5] Graubard, S. *The Artificial Intelligence Debate*, MIT Press, Cambridge, Mass. 1990, pp. 15-16.

[6] Hayes, P. *The Naive Physics Manifesto*, in Michie D. (Ed.), Expert Systems in the Microelectric Age, Edinburgh University Press, Edinburgh, 1979, pp. 242-270.

[7] Kosko, B. *Neural Networks and Fuzzy Systems: A Dynamical System Approach to Machine Intelligence*, Prentice Hall, Englewood Cliffs, New Jersey, 1992.

[8] Le Cun, Y. and E. Bienenstock, F. Souli, G. Weisbuch. *Learning Processes in An Asymmetric Threshold Networks*, Proceedings of Cognitive 85, Paris, France, pp. 599-604.

[9] Lippmann, R. *An Introduction to Computing with Neural Nets*, IEEE ASSP Magazine, April 1987, pp. 4-22.

[10] McClelland, J. and D. Rumelhart. *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercise*. MIT Press, London, England, 1988, pp. 135-137.

[11] McCluskey, E. *Logic Design Principles: with Emphasis on Testable Semicustom Circuits*, Prentice Hall, Englewood Cliffs, New Jersey, 1986, p. 41.

[12] McCulloch, W. and W. Pitts. *A Logical Calculus of the Ideas Immanent in Nervous Activity*, Bulletin of Mathematical Biophysics, Number 5, 1943, pp. 115-133.

[13] Parker, D. *Learning Logic*, Technical Report TR-47, Center for Computational Research in Economics and Management Science, MIT Press, Cambridge, Mass. 1985.

[14] Pedrycz, W. and A. Rocha. *Distributed Modeling*, Proceedings of 2nd International Conference on Fuzzy Logic and Neural Networks, Iizuka, Japan, Volume 1, July 17-22, 1992, pp. 217-220.

[15] Pedrycz, W. *Direct and Inverse Problem in Comparison of Fuzzy Data*, Fuzzy Sets and Systems, Volume 34, 1990, pp. 233-236.

[16] Pedrycz, W. *Fuzzy Control and Fuzzy Systems*, Research Studies Press, John Wiley, New York, 2nd edition, 1993, pp. 8-11.

[17] Pedrycz, W. *Fuzzy Neural Network with Reference Neurons as Pattern Classifiers*, IEEE Transactions on Neural Networks, Volume 3, Number 5, September 1992.

[18] Pedrycz, W. *Fuzzy Neural Networks and Neurocomputations*, Fuzzy Sets and Systems, 58, 1993, pp. 1-28.

[19] Pedrycz, W. *Lecture Notes for Current Research Topics in Computer Engineering*, University of Manitoba, 1992.

[20] Pedrycz, W. *Neurocomputations in Relational Systems*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 13, Number 3, 1991, pp. 289-296.

[21] Rescher, N. *Many-Valued Logic*, McGraw Hill, New York, 1969, pp. 22-42.

[22] Rumelhart, D. and G. Hinton, R. Williams. *Learning Internal Representations by Error Propagation*, Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Volume 1, MIT Press, Cambridge, Mass. 1986, pp. 318-362.

[23] Schwartz, D. and G. J. Klir. *Fuzzy Logic Flowers in Japan*, IEEE Spectrum, Volume 29, Number 7, July 1992, pp. 32-35.

[24] Silva, F. and L. Almeida. *Speeding Up Backpropagration*, Proceedings of Symposium on Neural Network for Sensory and Motor System, 1990.

[25] Werbos, P. *Beyond Regression: New Tools for Prediction and Analysis in Behavioral Science*, Ph. D. Thesis, Harvard University, Cambridge, Mass. 1974.

[26] Yager, R. and S. Ovchinnikov, R. Tong, H. Nguyen. (Eds.) *Fuzzy Sets and Applications: Selected Papers by L. A. Zadeh*, John Wiley, New York, 1987.

[27] Zadeh, L. A. *Fuzzy Sets*, Information and Control, Volume 8, 1965, pp. 338-353.

[28] Zadeh, L. *Outline of A New Approach to the Analysis of Complex Systems and Decision Processes*, IEEE Transactions on Systems, Man and Cybernetics. SMC-3, 1979, pp. 28-44.

Appendix A

A.1 On Line Learning Using Equality Index as Performance Index

The performance index used in optimization of network is given below:

$$Q_1 = (d_k \equiv y_k) = \begin{cases} 1 + d_k - y_k & \text{if } y_k > d_k \\ 1 - y_k + d_k & \text{if } y_k < d_k \\ 1 & \text{if } y_k = d_k \end{cases}$$

while the formulas describing the network are:

$$y_k = \overset{h}{\underset{i=1}{\mathrm{PS}}}\left(\mathrm{P}(w_i , z_i) \right)$$

$$z_i = \overset{m}{\underset{j=1}{\mathrm{P}}}\left(\mathrm{PS}(v_{ij} , x_{kj}) \right)$$

and the adjustments of the connections are governed by the expressions

$$\Delta w_c = \frac{\partial Q_1}{\partial w_c} = \frac{\partial Q_1}{\partial y_k} \cdot \frac{\partial y_k}{\partial w_c} \qquad c = 1, \ldots$$

$$\Delta v_{ab} = \frac{\partial Q_1}{\partial v_{ab}} = \frac{\partial Q_1}{\partial y_k} \cdot \frac{\partial y_k}{\partial z_a} \cdot \frac{\partial z_a}{\partial v_{ab}} \qquad a = 1, \ldots; \quad b = 1, \ldots$$

In what follows, we will provide complete formulas for determining Δw_c and Δv_{ab}.

A.1.1 Calculation of Δw_c

Bearing in mind the character of the performance index, we will study three separate cases: $y_k > d_k$, $y_k < d_k$, $y_k = d_k$.

Case i: $y_k > d_k$

$$\Delta w_c = \frac{\partial Q_1}{\partial w_c} = \frac{\partial Q_1}{\partial y_k} \cdot \frac{\partial y_k}{\partial w_c}$$

$$= \frac{\partial}{\partial y_k}(1 + d_k - y_k) \cdot \frac{\partial}{\partial w_c}\left[\overset{h}{\underset{i=1}{\mathrm{PS}}}\left(\mathrm{P}(w_i, z_i)\right)\right]$$

$$= (-1) \cdot \frac{\partial}{\partial w_c}\left\{\left[\overset{h}{\underset{i=1,i\neq c}{\mathrm{PS}}}\left(\mathrm{P}(w_i, z_i)\right)\right] \mathrm{PS}\left(\mathrm{P}(w_c, z_c)\right)\right\}$$

$$= -\frac{\partial}{\partial w_c}\left\{\mathrm{P}(w_c, z_c) + \overset{h}{\underset{i=1,i\neq c}{\mathrm{PS}}}\left(\mathrm{P}(w_i, z_i)\right) - \left[\mathrm{P}(w_c, z_c)\right] \cdot \left[\overset{h}{\underset{i=1,i\neq c}{\mathrm{PS}}}\left(\mathrm{P}(w_i, z_i)\right)\right]\right\}$$

$$= -\frac{\partial}{\partial w_c}\left\{(w_c \cdot z_c) + \overset{h}{\underset{i=1,i\neq c}{\mathrm{PS}}}(w_i \cdot z_i) - (w_c \cdot z_c) \cdot \left[\overset{h}{\underset{i=1,i\neq c}{\mathrm{PS}}}(w_i \cdot z_i)\right]\right\}$$

$$= -\left\{z_c - z_c \cdot \left[\overset{h}{\underset{i=1,i\neq c}{\mathrm{PS}}}(w_i \cdot z_i)\right]\right\}$$

$$= \left[\overset{h}{\underset{i=1,i\neq c}{\mathrm{PS}}}(w_i \cdot z_i) - 1\right] \cdot z_c$$

Case ii: $y_k < d_k$

The procedure for obtaining Δw_c for $y_k < d_k$ is similar to case i, and yields the final result in the form:

$$\Delta w_c = \left[1 - \overset{h}{\underset{i=1,i\neq c}{\mathrm{PS}}}(w_i \cdot z_i)\right] \cdot z_c$$

Case iii: $y_k = d_k$

$$\begin{aligned}\Delta w_c &= \frac{\partial Q_1}{\partial w_c} = \frac{\partial Q_1}{\partial y_k} \cdot \frac{\partial y_k}{\partial w_c} \\ &= \frac{\partial}{\partial y_k}(1) \cdot \frac{\partial y_k}{\partial w_c} \\ &= 0\end{aligned}$$

A.1.2 Calculation of Δv_{ab}

Again we will study three separate cases: $y_k > d_k$, $y_k < d_k$, $y_k = d_k$.

Case i: $y_k > d_k$

$$\begin{aligned}\Delta v_{ab} &= \frac{\partial Q_1}{\partial v_{ab}} = \frac{\partial Q_1}{\partial y_k} \cdot \frac{\partial y_k}{\partial z_a} \cdot \frac{\partial z_a}{\partial v_{ab}} \\ &= \frac{\partial}{\partial y_k}(1 + d_k - y_k) \cdot \frac{\partial}{\partial z_b}\left[\underset{i=1}{\overset{h}{\mathrm{PS}}}\left(\mathrm{P}(w_i, z_i)\right)\right] \cdot \frac{\partial}{\partial v_{ab}}\left[\underset{j=1}{\overset{m}{\mathrm{P}}}\left(\mathrm{PS}(v_{aj}, x_{kj})\right)\right] \\ &= (-1) \cdot \left[1 - \underset{i=1,i\neq a}{\overset{h}{\mathrm{PS}}}\left(\mathrm{P}(w_i, z_i)\right)\right] \cdot w_a \cdot \frac{\partial}{\partial v_{ab}}\left\{\left[\underset{j=1,j\neq b}{\overset{m}{\mathrm{P}}}\left(\mathrm{PS}(v_{aj}, x_{kj})\right)\right] \cdot \left(\mathrm{PS}(v_{ab}, x_{kb})\right)\right\} \\ &= (-1) \cdot \left[1 - \underset{i=1,i\neq a}{\overset{h}{\mathrm{PS}}}(w_i \cdot z_i)\right] \cdot w_a \cdot \frac{\partial}{\partial v_{ab}}\left\{\left[\underset{j=1,j\neq b}{\overset{m}{\mathrm{P}}}\left(\mathrm{PS}(v_{aj}, x_{kj})\right)\right] \cdot \left(\mathrm{PS}(v_{ab}, x_{kb})\right)\right\} \\ &= \left[\underset{i=1,i\neq a}{\overset{h}{\mathrm{PS}}}(w_i \cdot z_i) - 1\right] \cdot w_a \cdot \left[\underset{j=1,j\neq b}{\overset{m}{\mathrm{P}}}\left(\mathrm{PS}(v_{aj}, x_{kj})\right)\right] \cdot \frac{\partial}{\partial v_{ab}}(v_{ab} + x_{kb} - v_{ab} \cdot x_{kb}) \\ &= \left[\underset{i=1,i\neq a}{\overset{h}{\mathrm{PS}}}(w_i \cdot z_i) - 1\right] \cdot w_a \cdot \left[\underset{j=1,j\neq b}{\overset{m}{\mathrm{P}}}\left(\mathrm{PS}(v_{aj}, x_{kj})\right)\right] \cdot (1 - x_{kb})\end{aligned}$$

Case ii: $y_k < d_k$

The procedure for obtaining Δv_{ab} for $y_k < d_k$ is similar to case i, and the final result is in the form of:

$$\Delta v_{ab} = \left[1 - \underset{i=1,i\neq a}{\overset{h}{\mathrm{PS}}}\left(w_i, z_i\right)\right] \cdot w_a \cdot \left[\underset{j=1,j\neq b}{\overset{m}{\mathrm{P}}}\left(\mathrm{PS}\left(v_{aj}, x_{kj}\right)\right)\right] \cdot \left(1 - x_{kb}\right)$$

Case iii: $y_k = d_k$

$$\begin{aligned}
\Delta v_{ab} &= \frac{\partial Q_1}{\partial v_{ab}} = \frac{\partial Q_1}{\partial y_k} \cdot \frac{\partial y_k}{\partial z_a} \cdot \frac{\partial z_a}{\partial v_{ab}} \\
&= \frac{\partial}{\partial y_k}(1) \cdot \frac{\partial y_k}{\partial z_b} \cdot \frac{\partial z_b}{\partial v_{ab}} \\
&= 0 \cdot \frac{\partial y_k}{\partial z_b} \cdot \frac{\partial z_b}{\partial v_{ab}} \\
&= 0
\end{aligned}$$

A.2 On Line Learning Using Mean Square Error as Performance Index

The performance index used in optimization of network is given below:

$$Q_2 = \frac{1}{2}(y_k - d_k)^2$$

while the formulas describing the network are:

$$y_k = \overset{h}{\underset{i=1}{\mathrm{PS}}}\left(\mathrm{P}(w_i, z_i)\right)$$

$$z_i = \overset{m}{\underset{j=1}{\mathrm{P}}}\left(\mathrm{PS}(v_{ij}, x_{kj})\right)$$

and the adjustments of the connections are governed by the expressions:

$$\Delta w_c = \frac{\partial Q_2}{\partial w_c} = \frac{\partial Q_2}{\partial y_k}\cdot\frac{\partial y_k}{\partial w_c} \qquad c = 1, \ldots$$

$$\Delta v_{ab} = \frac{\partial Q_2}{\partial v_{ab}} = \frac{\partial Q_2}{\partial y_k}\cdot\frac{\partial y_k}{\partial z_a}\cdot\frac{\partial z_a}{\partial v_{ab}} \qquad a = 1, \ldots; \quad b = 1, \ldots$$

In what follows, we will provide complete formulas for determining Δw_c and Δv_{ab}.

A.2.1 Calculation of Δw_c

$$\begin{aligned}
\Delta w_c &= \frac{\partial Q_2}{\partial w_c} = \frac{\partial Q_2}{\partial y_k}\cdot\frac{\partial y_k}{\partial w_c} \\
&= \frac{\partial}{\partial y_k}\left(\frac{1}{2}(y_k - d_k)^2\right)\cdot\frac{\partial}{\partial w_c}\left[\overset{h}{\underset{i=1}{\mathrm{PS}}}\left(\mathrm{P}(w_i, z_i)\right)\right] \\
&= -(y_k - d_k)\cdot\frac{\partial}{\partial w_c}\left\{\left[\overset{h}{\underset{i=1,i\neq c}{\mathrm{PS}}}\left(\mathrm{P}(w_i, z_i)\right)\right]\mathrm{PS}\left(\mathrm{P}(w_c, z_c)\right)\right\} \\
&= (d_k - y_k)\cdot\frac{\partial}{\partial w_c}\left\{\mathrm{P}(w_c, z_c) + \overset{h}{\underset{i=1,i\neq c}{\mathrm{PS}}}\left(\mathrm{P}(w_i, z_i)\right) - \left[\mathrm{P}(w_c, z_c)\right]\cdot\left[\overset{h}{\underset{i=1,i\neq c}{\mathrm{PS}}}\left(\mathrm{P}(w_i, z_i)\right)\right]\right\}
\end{aligned}$$

$$= (d_k - y_k) \cdot \frac{\partial}{\partial w_c} \left\{ w_c \cdot z_c + \underset{i=1, i \neq c}{\overset{h}{\mathrm{PS}}} (w_i \cdot z_i) - (w_c \cdot z_c) \cdot \left[\underset{i=1, i \neq c}{\overset{h}{\mathrm{PS}}} (w_i \cdot z_i) \right] \right\}$$

$$= (d_k - y_k) \cdot \left\{ z_c - z_c \cdot \left[\underset{i=1, i \neq c}{\overset{h}{\mathrm{PS}}} (w_i \cdot z_i) \right] \right\}$$

$$= (d_k - y_k) \cdot \left[1 - \underset{i=1, i \neq c}{\overset{h}{\mathrm{PS}}} (w_i \cdot z_i) \right] \cdot z_c$$

A.2.2 Calculation of Δv_{ab}

$$\Delta v_{ab} = \frac{\partial Q_2}{\partial v_{ab}} = \frac{\partial Q_2}{\partial y_k} \cdot \frac{\partial y_k}{\partial z_a} \cdot \frac{\partial z_a}{\partial v_{ab}}$$

$$= \frac{\partial}{\partial y_k} \left(\frac{1}{2} (y_k - d_k)^2 \right) \cdot \frac{\partial}{\partial z_b} \left[\underset{i=1}{\overset{h}{\mathrm{PS}}} \left(\mathrm{P}(w_i, z_i) \right) \right] \cdot \frac{\partial}{\partial v_{ab}} \left[\underset{j=1}{\overset{m}{\mathrm{P}}} \left(\mathrm{PS}(v_{aj}, x_{kj}) \right) \right]$$

$$= -(y_k - d_k) \cdot \left[1 - \underset{i=1, i \neq a}{\overset{h}{\mathrm{PS}}} \left(\mathrm{P}(w_i, z_i) \right) \right] \cdot w_a \cdot \frac{\partial}{\partial v_{ab}} \left\{ \left[\underset{j=1, j \neq b}{\overset{m}{\mathrm{P}}} \left(\mathrm{PS}(v_{aj}, x_{kj}) \right) \right] \cdot \left(\mathrm{PS}(v_{ab}, x_{kb}) \right) \right\}$$

$$= -(y_k - d_k) \cdot \left[1 - \underset{i=1, i \neq a}{\overset{h}{\mathrm{PS}}} (w_i \cdot z_i) \right] \cdot w_a \cdot \frac{\partial}{\partial v_{ab}} \left\{ \left[\underset{j=1, j \neq b}{\overset{m}{\mathrm{P}}} \left(\mathrm{PS}(v_{aj}, x_{kj}) \right) \right] \cdot \left(\mathrm{PS}(v_{ab}, x_{kb}) \right) \right\}$$

$$= (d_k - y_k) \cdot \left[1 - \underset{i=1, i \neq a}{\overset{h}{\mathrm{PS}}} (w_i \cdot z_i) \right] \cdot w_a \cdot \left[\underset{j=1, j \neq b}{\overset{m}{\mathrm{P}}} \left(\mathrm{PS}(v_{aj}, x_{kj}) \right) \right] \cdot \frac{\partial}{\partial v_{ab}} (v_{ab} + x_{kb} - v_{ab} \cdot x_{kb})$$

$$= (d_k - y_k) \cdot \left[1 - \underset{i=1, i \neq a}{\overset{h}{\mathrm{PS}}} (w_i \cdot z_i) \right] \cdot w_a \cdot \left[\underset{j=1, j \neq b}{\overset{m}{\mathrm{P}}} \left(\mathrm{PS}(v_{aj}, x_{kj}) \right) \right] \cdot (1 - x_{kb})$$

Appendix B

B.1 Header for Program 'Logic_Processor.c'

```
/* Logic_Processor.h -- constants and declarations for Logic_Processor.c */

/*_____________________________________________________________*/
/*                                                             */
/*                            by                               */
/*                                                             */
/*                        Patrick LAM                          */
/*                                                             */
/*          Department of Electrical & Computer Engineering    */
/*                    The University of Manitoba               */
/*                      Winnipeg, Manitoba                     */
/*                          R3T 2N2                            */
/*                                                             */
/*                        Version 1.0                          */
/*                        Jan 21, 1993                         */
/*                                                             */
/*_____________________________________________________________*/

#define numOfData 250              /* number of tested data */
#define numOfHidden 8              /* number of elements in the hidden layer */
#define numOfInput 3               /* number of external input to the logic processor */
#define numOfStateVar 3            /* number of state variables to the logic processor */
#define totalInput (2*(numOfInput + numOfStateVar))        /* total number of input to the LP */
#define Z 0.8                      /* momentum term in the update procedure */
#define steps 50000                /* number of learning steps */
#define same 1.00                  /* the updating step size parameters for the same sign */
#define opposite 1.00              /* the updating step size parameters for the opposite sign */
#define C 3.0                      /* the degree of the function */
#define gain1 0.1
#define AccLevelError 0.000001     /* the acceptable level of error */
#define transfer 0                 /* transfer=1, the transfer fcn at the output node is enable */
#define trainingData "mxor.dat"
#define randNum 54
#define upperLimit 1000
#define stars "**********************************************************"
```

```
FILE    *infile;                    /* set input, output file pointer */

void    init(double input[][totalInput], double tarOp[], double epsilon, double v[][totalInput],
            double newV[][totalInput], double w[], double newW[], double newDv[][totalInput],
            double oldDv[][totalInput], double oldLRV[][totalInput], double newLRV[][totalInput],
            double newDw[], double oldDw[], double oldLRW[], double newLRW[]);

void    initData(double input[][totalInput], double tarOp[]);

void    initWeight(double epsilon, double v[][totalInput], double newV[][totalInput], double w[],
            double newW[]);

void    initDelta(double newDv[][totalInput], double oldDv[][totalInput], double oldLRV[][totalInput],
            double newLRV[][totalInput], double newDw[], double oldDw[], double oldLRW[],
            double newLRW[]);

void    process(long int n, double pArry[], double *minError, double w[], double minW[],
            double v[][totalInput], double minV[][totalInput], double input[][totalInput],
            double hidden[][numOfHidden], double yout[], double output[], double tarOp[],
            double newDw[], double oldDw[], double newLRW[], double oldLRW[], double changeW[],
            double newW[], double newDv[][totalInput], double oldDv[][totalInput],
            double newLRV[][totalInput], double oldLRV[][totalInput], double changeV[][totalInput],
            double newV[][totalInput], double epsilon);

void    updateMinWei(double w[], double minW[], double v[][totalInput], double minV[][totalInput]);

void    updateOutput(double v[][totalInput], double input[][totalInput], double hidden[][numOfHidden],
            double w[], double yout[], double output[]);

void    hiddenElement(int counter, double v[][totalInput], double input[][totalInput],
            double hidden[][numOfHidden]);

void    outputElement(int counter, double w[], double hidden[][numOfHidden], double yout[],
            double output[]);

double  transferFcn(double x);

double  tNorm(double x, double y);

double  sNorm(double x, double y);

double  performance(double tarOp[], double yout[]);

void    learning(long int k, double newDw[], double hidden[][numOfHidden], double yout[],
            double tarOp[], double output[], double w[], double pArry[], double oldDw[],
            double newLRW[], double oldLRW[], double changeW[], double newW[], double epsilon,
            double newDv[][totalInput], double oldDv[][totalInput], double newLRV[][totalInput],
            double oldLRV[][totalInput], double changeV[][totalInput], double v[][totalInput],
            double newV[][totalInput], double input[][totalInput]);
```

```
void    findDeltaW(double newDw[], double hidden[][numOfHidden], double yout[], double tarOp[],
            double output[], double w[]);

double  deltaW(int counter, double hidden[][numOfHidden], double yout[], double tarOp[],
            double output[], double w[]);

double  deltaY(int counter, int i, double w[], double hidden[][numOfHidden]);

double  devFunction(double y);

int     wCond(int counter, double w[]);

void    findNewW(long int k, double pArry[], double newDw[], double oldDw[], double newLRW[],
            double oldLRW[], double changeW[], double w[], double newW[], double epsilon);

int     checkSign(double x, double y);

double  fun(long int k);

void    findDeltaV(double newDv[][totalInput], double tarOp[], double yout[], double w[],
            double output[], double v[][totalInput], double input[][totalInput],
            double hidden[][numOfHidden]);

double  deltaV(int i, int j, double tarOp[], double yout[], double w[], double output[],
            double v[][totalInput], double input[][totalInput], double hidden[][numOfHidden]);

double  deltaHidden(int i, int j, int k, double v[][totalInput], double input[][totalInput], double w[],
            double hidden[][numOfHidden]);

int     vCond(int i, int j, int k, double w[], double hidden[][numOfHidden]);

void    findNewV(long int k, double pArry[], double newDv[][totalInput], double oldDv[][totalInput],
            double newLRV[][totalInput], double oldLRV[][totalInput], double changeV[][totalInput],
            double v[][totalInput], double newV[][totalInput], double epsilon);

void    updateW(double w[], double newW[]);

void    updateV(double v[][totalInput], double newV[][totalInput]);

void    updateDeltaW(double oldDw[], double newDw[]);

void    updateDeltaV(double oldDv[][totalInput], double newDv[][totalInput]);

void    outputResult(long int k, double pArry[], double w[], double v[][totalInput], double minError,
            double minW[], double minV[][totalInput], double input[][totalInput],
            double tarOp[], double yout[]);

void    prnIndex(double pArry[], long int k);

void    prtOutput(double w[], double v[][totalInput]);

void    prnt(double input[][totalInput], double tarOp[], double yout[]);
```

```
void    prtMin(double minError, double minW[], double minV[][totalInput]);

void    prtMinW(double minW[], double minV[][totalInput]);

void    prtOutputW(double w[], double v[][totalInput]);
```

B.2 Program Listing for 'Logic_Processor.c'

```
#include <stdio.h>
#include <math.h>
#include <time.h>
#include "Logic_Processor.h"

/*_____________________________________________________________*/
/*                                                             */
/*                      LOGIC  PROCESSOR                       */
/*                                                             */
/*_____________________________________________________________*/
/*                                                             */
/*                           by                                */
/*                                                             */
/*                       Patrick LAM                           */
/*                                                             */
/*          Department of Electrical & Computer Engineering    */
/*                    The University of Manitoba               */
/*                      Winnipeg, Manitoba                     */
/*                         R3T 2N2                             */
/*                                                             */
/*                       Version 1.0                           */
/*                       Jan 21, 1993                          */
/*                                                             */
/*_____________________________________________________________*/
/*                                                             */
/*    This program uses the mean square error to justify the performance of  */
/*    the network, P.S. and Product are used for s- and t-norm.  Off-line    */
/*    learning techique is applied.                            */
/*                                                             */
/*                                                             */
/*_____________________________________________________________*/

int     main(void)
```

```
{
        double    input[numOfData][totalInput];          /* the input to the LP */
        double    tarOp[numOfData];                      /* target output of the LP */
        double    output[numOfData];                     /* actural output of the LP */
        double    yout[numOfData];                       /* actural output of the LP */
        double    pArry[steps];                          /* the performance indices */
        double    hidden[numOfData][numOfHidden];        /* the results of the elements in the hidden *
                                                         * layer */
        double    w[numOfHidden];                        /* the weights of the element in the hidden *
                                                         * layer */
        double    v[numOfHidden][totalInput];            /* the weights of the elements in the *
                                                         * input layer */
        double    newW[numOfHidden];                     /* the new weights of the elements in the*
                                                         * hidden layer */
        double    newV[numOfHidden][totalInput];         /* the new weights of the elements in *
                                                         * the input layer */
        double    minW[numOfHidden];                     /* the weight for min. performance index */
        double    minV[numOfHidden][totalInput];         /* the weights for min. performance index */
        double    changeW[numOfHidden];                  /* the changes of the weights in the *
                                                         * hidden layer */
        double    changeV[numOfHidden][totalInput];      /* the changes of the weights in the input *
                                                         * layer */
        double    newDw[numOfHidden];                    /* the new values for the delta weights in *
                                                         * the hidden layer */
        double    oldDw[numOfHidden];                    /* the old values for the delta weights in *
                                                         * the hidden layer */
        double    newDv[numOfHidden][totalInput];        /* the new values for the delta weights in *
                                                         * the input layer */
        double    oldDv[numOfHidden][totalInput];        /* the old values for the delta weights in *
                                                         * the input layer */
        double    oldLRW[numOfHidden];                   /* the old learning rate for the W */
        double    newLRW[numOfHidden];                   /* the new learning rate for the W */
        double    oldLRV[numOfHidden][totalInput];       /* the old learning rate for the V */
        double    newLRV[numOfHidden][totalInput];       /* the new learning rate for the V */
        double    minError;                              /* the minimum error so far occured in the*
                                                         * process */

        double    epsilon;
        long int  k;                                     /* counter for learning steps */
        double    TargetError;                           /* the error level = max E.I. - *
                                                         * AccLevelError */
```

```
/*||_____________________________________________________________||*/
/*||_____________________________________________________________||*/
/*||                                                             ||*/
/*||                        main  program                        |*/
/*||                                                             ||*/
/*||_____________________________________________________________||*/
/*||_____________________________________________________________||*/
```

```
  k = 0;
  srand(randNum);
```

```
    epsilon = fun(k);

        /* this procedure call initData, initWeight, initDelta */
    init(input, tarOp, epsilon, v, newV, w, newW, newDv, oldDv, oldLRV, newLRV, newDw,
        oldDw, oldLRW, newLRW);

    TargetError = AccLevelError;
    pArry[0] = upperLimit;
    while ((k < steps) && (pArry[k] > TargetError))
    {
                /* perform an learning iteration */
        k = k + 1;
        epsilon = fun(k);
        process(k, pArry, &minError, w, minW, v, minV, input, hidden, yout, output, tarOp, newDw,
                oldDw, newLRW, oldLRW, changeW, newW, newDv, oldDv, newLRV, oldLRV,
                changeV, newV, epsilon);
    }
    printf("learning stops at %ld., and P.I. = %lf.\n", k, pArry[k]);
    outputResult(k, pArry, w, v, minError, minW, minV, input, tarOp, yout);
    return 0;
}



/*_____________________________________________________________*/
/*                          init()                             */
/*_____________________________________________________________*/
/*   This procedure calls initData, initWeight, and initDalta. */
/*_____________________________________________________________*/

void   init(double input[][totalInput], double tarOp[], double epsilon, double v[][totalInput],
          double newV[][totalInput], double w[], double newW[], double newDv[][totalInput],
          double oldDv[][totalInput], double oldLRV[][totalInput], double newLRV[][totalInput],
          double newDw[], double oldDw[], double oldLRW[], double newLRW[])
{

        /* load the data set for the logic processor */
      initData(input, tarOp);

        /* init the weights in the logic processor */
      initWeight(epsilon, v, newV, w, newW);

      initDelta(newDv, oldDv, oldLRV, newLRV, newDw, oldDw, oldLRW, newLRW);
}



/*_____________________________________________________________*/
/*                          initData()                         */
/*_____________________________________________________________*/
/*   This procedure ininializes the data set, which includes the input data, target */
```

```
/*    output data for the LP.                                          */
/*_____________________________________________________________*/

void   initData(double input[][totalInput], double tarOp[])
{
       int     i, j;

       if ((infile = fopen(trainingData, "r")) == NULL)
       {
              printf("fopen failed.\n");
              exit(0);
       }
       for (i = 0; i < numOfData; i++)
       {
              for (j = 0; j < totalInput; j++)
              {
                     fscanf(infile, "%lf", &input[i][j]);
              }
              fscanf(infile, "%lf", &tarOp[i]);
       }
       fclose(infile);
}



/*_____________________________________________________________*/
/*                          initWeight()                        */
/*_____________________________________________________________*/
/*    This procedure initializes all the weights in the LP.     */
/*_____________________________________________________________*/

void   initWeight(double epsilon, double v[][totalInput], double newV[][totalInput], double w[],
         double newW[])
{
       int            i, j;
       double         temp;

       for (i = 0; i < numOfHidden; i++)
       {
              temp = rand() / pow(2.0, 15.0);
              if (temp < epsilon)
                     temp = epsilon;
              else if (temp > (1 - epsilon))
                     temp = 1 - epsilon;
              w[i] = temp;
              newW[i] = 0.0;
              for (j = 0; j < totalInput; j++)
              {
                     temp = rand() / pow(2.0, 15.0);
                     if (temp < epsilon)
                            temp = epsilon;
                     else if (temp > (1 - epsilon))
```

```
                        temp = 1 - epsilon;
                    v[i][j] = temp;
                    newV[i][j] = 0.0;
                }
        }
}


/*_____________________________________________________________*/
/*                        initDelta()                          */
/*_____________________________________________________________*/
/*  This procedure initializes the delta values of the weights  in the LP, it also  */
/*  initializes the learning rates.                            */
/*_____________________________________________________________*/

void   initDelta(double newDv[][totalInput], double oldDv[][totalInput], double oldLRV[][totalInput],
         double newLRV[][totalInput], double newDw[], double oldDw[], double oldLRW[],
         double newLRW[])
{
        int             i, j;
        double          temp;

        for (i = 0; i < numOfHidden; i++)
        {
                newDw[i] = 0.0;
                oldDw[i] = 0.0;
                newLRW[i] = 0.0;
                temp = rand() / pow(2.0, 15.0);
                if (temp > 1)
                        oldLRW[i] = 1.0;
                else
                        oldLRW[i] = temp;
                for (j = 0; j < totalInput; j++)
                {
                        newDv[i][j] = 0.0;
                        oldDv[i][j] = 0.0;
                        newLRV[i][j] = 0.0;
                        temp = rand() / pow(2.0, 15.0);
                        if (temp > 1)
                                oldLRV[i][j] = 1.0;
                        else
                                oldLRV[i][j] = temp;
                }
        }
}


/*_____________________________________________________________*/
/*                        process()                            */
```

```
/*_____________________________________________________________*/
/*    This procedure performs an learning iteration.           */
/*_____________________________________________________________*/

void  process(long int n, double pArry[], double *minError, double w[], double minW[],
        double v[][totalInput], double minV[][totalInput], double input[][totalInput],
        double hidden[][numOfHidden], double yout[], double output[], double tarOp[], double newDw[],
        double oldDw[], double newLRW[], double oldLRW[], double changeW[],
        double newW[], double newDv[][totalInput], double oldDv[][totalInput],
        double newLRV[][totalInput], double oldLRV[][totalInput], double changeV[][totalInput],
        double newV[][totalInput], double epsilon)
{
      updateOutput(v, input, hidden, w, yout, output);
      pArry[n] = performance(tarOp, yout);          /* find out the performance index of the LP */
      if (n == 1)
      {
            *minError = upperLimit;
      }
      else
      {
            if (pArry[n] < *minError)
            {
                  *minError = pArry[n];
                  updateMinWei(w, minW, v, minV);
            }
      }

            /* performs learning from the training set */
      learning(n, newDw, hidden, yout, tarOp, output, w, pArry, oldDw, newLRW, oldLRW, changeW,
            newW, epsilon, newDv, oldDv, newLRV, oldLRV, changeV, v, newV, input);
}


void  updateMinWei(double w[], double minW[], double v[][totalInput], double minV[][totalInput])

{
      int     i, j;

      for (i = 0; i < numOfHidden; i++)
      {
            minW[i] = w[i];
            for (j = 0; j < totalInput; j++)
            {
                  minV[i][j] = v[i][j];
            }
      }
}


/*_____________________________________________________________*/
```

```
/*                              updateOutput()                                  */
/*______________________________________________________________________________*/
/*      This procedure updates the output of the logic processor.               */
/*______________________________________________________________________________*/

void   updateOutput(double v[][totalInput], double input[][totalInput], double hidden[][numOfHidden],
         double w[], double yout[], double output[])
{
       int     counter;

       for (counter = 0; counter < numOfData; counter++)
       {
              hiddenElement(counter, v, input, hidden);
              outputElement(counter, w, hidden, yout, output);
       }
}



/*______________________________________________________________________________*/
/*                              hiddenElement()                                 */
/*______________________________________________________________________________*/
/*      This procedure calculates the results in the hidden layer.  The job of the */
/*      hidden layer is to make up the product term of the input.               */
/*______________________________________________________________________________*/

void   hiddenElement(int counter, double v[][totalInput], double input[][totalInput],
         double hidden[][numOfHidden])
{
       int     i, j;
       double        temp;

       for (i = 0; i < numOfHidden; i++)
       {
              temp = sNorm(v[i][0], input[counter][0]);
              for (j = 1; j < totalInput; j++)
              {
                     temp = tNorm(temp, (sNorm(v[i][j], input[counter][j])));
              }
              hidden[counter][i] = temp;
       }
}



/*______________________________________________________________________________*/
/*                              outputElement()                                 */
/*______________________________________________________________________________*/
/*      This procedure calculates the result in the output layer of Logic processor. */
/*______________________________________________________________________________*/
```

```
void   outputElement(int counter, double w[], double hidden[][numOfHidden], double yout[],
         double output[])
{
        int     i;
        double          temp;

        temp = tNorm(hidden[counter][0], w[0]);
        for (i = 1; i < numOfHidden; i++)
        {
                temp = sNorm(temp, (tNorm(hidden[counter][i], w[i])));
        }
        if (transfer)
        {
                yout[counter] = transferFcn(temp);
                output[counter] = temp;
        }
        else
        {
                yout[counter] = temp;
                output[counter] = temp;
        }
}


/*_____________________________________________________________*/
/*                          transferFcn()                      */
/*_____________________________________________________________*/
/*      This procedure returns the result of the transfer function.  */
/*_____________________________________________________________*/

double  transferFcn(double x)
{
        double      result, temp1, temp2;

        temp1 = -C;
        temp2 = temp1 * (x - 0.5);
        result = (1 / (1.0 + exp(temp2)));
        return result;
}


/*_____________________________________________________________*/
/*                          tNorm()                            */
/*_____________________________________________________________*/
/*      This procedure defines a t-norm.                       */
/*_____________________________________________________________*/

double          tNorm(double x, double y)
{
```

```
        double      temp;

        temp = x * y;
        return temp;
}


/*_____________________________________________________________*/
/*                          sNorm()                            */
/*_____________________________________________________________*/
/*      This procedure defines a s-norm.                       */
/*_____________________________________________________________*/

double      sNorm(double x, double y)
{
        double      temp;

        temp = x + y - x * y;
        return temp;
}


/*_____________________________________________________________*/
/*                        performance()                        */
/*_____________________________________________________________*/
/*  This is a function which returns the performace index of the logic processor.  */
/*_____________________________________________________________*/

double      performance(double tarOp[], double yout[])
{
        int     ;
        double      temp, sum, temp1;

        sum = 0.0;
        for (i = 0; i < numOfData; i++)
        {
                temp = (tarOp[i] - yout[i]);
                temp1 = temp * temp;
                sum += temp1;
        }
        sum *= 0.5;
        return sum;
}


/*_____________________________________________________________*/
/*                          learning()                         */
/*_____________________________________________________________*/
```

```
/*      This procedure performs the learning procedure.                    */
/*_____________________________________________________________________*/

void   learning(long int k, double newDw[], double hidden[][numOfHidden], double yout[],
         double tarOp[], double output[], double w[], double pArry[], double oldDw[], double newLRW[],
         double oldLRW[], double changeW[], double newW[], double epsilon,
         double newDv[][totalInput], double oldDv[][totalInput], double newLRV[][totalInput],
         double oldLRV[][totalInput], double changeV[][totalInput], double v[][totalInput],
         double newV[][totalInput], double input[][totalInput])
{
        /* find out the delta weights in the hidden layer */
      findDeltaW(newDw, hidden, yout, tarOp, output, w);

        /* calculate the new weights in the hidden layer */
      findNewW(k, pArry, newDw, oldDw, newLRW, oldLRW, changeW, w, newW, epsilon);

        /* find out the delta weights in the input layer */
      findDeltaV(newDv, tarOp, yout, w, output, v, input, hidden);

        /* calculate the new weights in the input layer */
      findNewV(k, pArry, newDv, oldDv, newLRV, oldLRV, changeV, v, newV, epsilon);

      updateW(w, newW);              /* update the weights in the hidden layer */

      updateV(v, newV);              /* update teh weights in the input layer */

      updateDeltaW(oldDw, newDw);    /* update the delta weights in the hidden layer */
      updateDeltaV(oldDv, newDv);    /* update the delta weights in the input layer */
}


/*_____________________________________________________________________*/
/*                          findDeltaW()                                */
/*_____________________________________________________________________*/
/*      This procedure finds the delta values of the weights in the hiddeen layer */
/*      of the logic processor.                                        */
/*_____________________________________________________________________*/

void   findDeltaW(double newDw[], double hidden[][numOfHidden], double yout[], double tarOp[],
         double output[], double w[])
{
      int     i;

      for (i = 0; i < numOfHidden; i++)
      {
            newDw[i] = deltaW(i, hidden, yout, tarOp, output, w);
      }
}
```

```
/*_____________________________________________________________*/
/*                          deltaW()                           */
/*_____________________________________________________________*/
/*  This procedure finds the delta values of the weights in the hidden layer of  */
/*  the logic processor.                                       */
/*_____________________________________________________________*/

double      deltaW(int counter, double hidden[][numOfHidden], double yout[], double tarOp[],
                double output[], double w[])
{
    int     i;
    double      ch, temp, temp1;

    temp = 0.0;
    for (i = 0; i < numOfData; i++)
    {
        temp1 = deltaY(counter, i, w, hidden) * hidden[i][counter] *
        (tarOp[i] - yout[i]);
        if (transfer)
            temp1 *= devFunction(output[i]);
        ch = temp1;
        temp += ch;
    }
    return temp;
}


/*_____________________________________________________________*/
/*                          deltaY()                           */
/*_____________________________________________________________*/
/*  This procedure finds the derivative of output Y with respect to the product  */
/*   of w[counter] and z[counter].                             */
/*_____________________________________________________________*/

double      deltaY(int counter, int i, double w[], double hidden[][numOfHidden])
{
    int         j;
    double      temp, result;

    if (numOfHidden == 1)
        result = 1.0;
    else
    {
        if (numOfHidden == 2)
        {
            if (counter == 0)
                result = 1 - sNorm((tNorm(w[1], hidden[i][1])), 0);
            else
                result = 1 - sNorm((tNorm(w[0], hidden[i][0])), 0);
        }
```

```
            else
            {
                if (counter != 0)
                {
                    temp = tNorm(w[0], hidden[i][0]);
                    for (j = 1; j < numOfHidden; j++)
                    {
                        if (j != counter)
                            temp = sNorm((tNorm(w[j], hidden[i][j])), temp);
                    }
                }
                else
                {
                    temp = tNorm(w[1], hidden[i][1]);
                    for (j = 2; j < numOfHidden; j++)
                    {
                        if (j != counter)
                            temp = sNorm((tNorm(w[j], hidden[i][j])), temp);
                    }
                }
                result = 1.0 - temp;
            }
        }
        return result;
}


/*_____________________________________________________________*/
/*                        devFunction()                        */
/*_____________________________________________________________*/
/*  This procedure finds the derivates of the transfer function.*/
/*_____________________________________________________________*/

double      devFunction(double y)
{
        double      result, temp;

        temp = C;
        result = transferFcn(y) * (1 - transferFcn(y));
        return result;
}


/*_____________________________________________________________*/
/*                          wCond()                            */
/*_____________________________________________________________*/
/*  This procedure checks if the conditions for calculating the values of the */
/*  delta of the weight in the hidden layer is satisfied.      */
/*_____________________________________________________________*/
```

```
int     wCond(int counter, double w[])
{

        if (w[counter] != 0)
                return 1;
        else
                return 0;
}



/*_____________________________________________________________*/
/*                          findNewW()                         */
/*_____________________________________________________________*/
/*  This procedure calculates the new weights in the hidden layer of the logic  */
/*  processor.                                                 */
/*_____________________________________________________________*/

void    findNewW(long int k, double pArry[], double newDw[], double oldDw[], double newLRW[],
          double oldLRW[], double changeW[], double w[], double newW[], double epsilon)
{
        int     counter;
        double          temp;
        double          error;

        error = gain1 * (pArry[k] - pArry[k - 1]) / pArry[k];

        for (counter = 0; counter < numOfHidden; counter++)
        {
                if (newDw[counter] == 0 || oldDw[counter] == 0)
                        newLRW[counter] = oldLRW[counter];
                else
                {
                        if (checkSign(oldDw[counter], newDw[counter]))
                        {
                                if (new)
                                        newLRW[counter] = (1.0 + error) * oldLRW[counter];
                                else
                                        newLRW[counter] = same * oldLRW[counter];
                        }
                        else
                        {
                                if (new)
                                        newLRW[counter] = (1.0 - error) * oldLRW[counter];
                                else
                                        newLRW[counter] = opposite * oldLRW[counter];
                        }
                }

                changeW[counter] = newLRW[counter] * ((newDw[counter] / numOfData) +
                        (Z * oldDw[counter] / numOfData));
```

```
                temp = w[counter] + changeW[counter];
                if (temp < epsilon)
                        newW[counter] = epsilon;
                else if (temp > (1 - epsilon))
                        newW[counter] = 1 - epsilon;
                else
                        newW[counter] = temp;
                oldLRW[counter] = newLRW[counter];
        }
}


/*_____________________________________________________________*/
/*                          checkSign()                        */
/*_____________________________________________________________*/
/*  This procedure checks if the two given inputs have the same sign.  */
/*_____________________________________________________________*/

int     checkSign(double x, double y)
{
        if ((x > 0 && y > 0) || (x < 0 && y < 0))
                return 1;
        else
                return 0;
}


/*_____________________________________________________________*/
/*                          fun()                              */
/*_____________________________________________________________*/
/*  This is a decreasing function for calculating the effects of the weights in  */
/*  the LP.                                                    */
/*_____________________________________________________________*/

double          fun(long int k)
{
        int     l;
        double          temp, x, y;

        x = k;
        l = 2;
        y = 3.0 / l;
        if (k > 0)
                temp = (1 / ((exp(y * log(x))) + 25.0));
        else
                temp = 1.0 / 25.0;
        return temp;
}
```

```
/*_____________________________________________________________*/
/*                          findDeltaV()                       */
/*_____________________________________________________________*/
/*    This procedure finds the delta values of the weights in the input layer of  */
/*    the logic processor.                                     */
/*_____________________________________________________________*/

void  findDeltaV(double newDv[][totalInput], double tarOp[], double yout[], double w[],
        double output[], double v[][totalInput], double input[][totalInput],
        double hidden[][numOfHidden])
{
      int    i, j;

      for (i = 0; i < numOfHidden; i++)
      {
            for (j = 0; j < totalInput; j++)
            {
                  newDv[i][j] = deltaV(i, j, tarOp, yout, w, output, v, input, hidden);
            }
      }
}


/*_____________________________________________________________*/
/*                          deltaV()                           */
/*_____________________________________________________________*/
/*    This procedure finds the delta values of the weights in the input layer of  */
/*    the logic processor.                                     */
/*_____________________________________________________________*/

double  deltaV(int i, int j, double tarOp[], double yout[], double w[], double output[],
        double v[][totalInput], double input[][totalInput], double hidden[][numOfHidden])
{
      int    k;
      double       ch, temp, temp1;

      temp = 0;
      for (k = 0; k < numOfData; k++)
      {
            temp1 = (tarOp[k] - yout[k]) * w[i] * deltaY(i, k, w, hidden) *
            deltaHidden(i, j, k, v, input, w, hidden) *  (1 - input[k][j]);
            if (transfer)
                  temp1 *= devFunction(output[k]);
            ch = temp1;
            temp += ch;
      }
      return temp;
}
```

```
/*_____________________________________________________________*/
/*                        deltaHidden()                        */
/*_____________________________________________________________*/
/*  This procedure finds the derivative of z(counter) with respect to the  */
/*  weight between the input and hidden layers.                */
/*_____________________________________________________________*/

double      deltaHidden(int i, int j, int k, double v[][totalInput], double input[][totalInput], double w[],
                double hidden[][numOfHidden])
{
        int             count;
        double          temp;

        if (j != 0)
        {
                temp = sNorm(v[i][0], input[k][0]);
                for (count = 1; count < totalInput; count++)
                {
                        if (count != j)
                        temp = tNorm((sNorm(v[i][count], input[k][count])), temp);
                }
        }
        else
        {
                temp = sNorm(v[i][1], input[k][1]);
                for (count = 2; count < totalInput; count++)
                {
                        temp = tNorm((sNorm(v[i][count], input[k][count])), temp);
                }
        }
        return temp;
}


/*_____________________________________________________________*/
/*                          vCond()                            */
/*_____________________________________________________________*/
/*  This procedure checks if the conditions for calculating the values of the  */
/*  delta of the weight in the input layer is satisfied.       */
/*_____________________________________________________________*/

int     vCond(int i, int j, int k, double w[], double hidden[][numOfHidden])
{
        if ((w[i] != 0) && (hidden[k][i] != 0))
                return 1;
        else
                return 0;
```

```
}


/*_____________________________________________________________*/
/*                          findNewV()                         */
/*_____________________________________________________________*/
/*  This procedure calculates the new weights in the input layer of the logic  */
/*  processor.                                                 */
/*_____________________________________________________________*/

void  findNewV(long int k, double pArry[], double newDv[][totalInput], double oldDv[][totalInput],
        double newLRV[][totalInput], double oldLRV[][totalInput], double changeV[][totalInput],
        double v[][totalInput], double newV[][totalInput], double epsilon)
{
      int    i, j;
      double      temp;
      double      error;

      error = gain1 * (pArry[k] - pArry[k - 1]) / pArry[k];
      for (i = 0; i < numOfHidden; i++)
      {
            for (j = 0; j < totalInput; j++)
            {
                  if (newDv[i][j] == 0 || oldDv[i][j] == 0)
                        newLRV[i][j] = oldLRV[i][j];
                  else
                  {
                        if (checkSign(oldDv[i][j], newDv[i][j]))
                        {
                              if (new)
                                    newLRV[i][j] = (1.0 + error) * oldLRV[i][j];
                              else
                                    newLRV[i][j] = same * oldLRV[i][j];
                        }
                        else
                        {
                              if (new)
                                    newLRV[i][j] = (1.0 - error) * oldLRV[i][j];
                              else
                                    newLRV[i][j] = opposite * oldLRV[i][j];
                        }
                  }

                  changeV[i][j] = newLRV[i][j] * ((newDv[i][j] / numOfData) +
                        (Z * oldDv[i][j] / numOfData));
                  temp = v[i][j] + changeV[i][j];

                  if (temp < epsilon)
                        newV[i][j] = epsilon;
                  else if (temp > (1 - epsilon))
                        newV[i][j] = 1 - epsilon;
```

```
                else
                        newV[i][j] = temp;
                oldLRV[i][j] = newLRV[i][j];
            }
        }
}


/*_____________________________________________________________*/
/*                          updateW()                          */
/*_____________________________________________________________*/
/*  This procedure update the weights in the hidden layer of the logic  */
/*  processor.                                                 */
/*_____________________________________________________________*/

void    updateW(double w[], double newW[])
{
        int     i;

        for (i = 0; i < numOfHidden; i++)
        {
                w[i] = newW[i];
        }
}


/*_____________________________________________________________*/
/*                          updateV()                          */
/*_____________________________________________________________*/
/*  This procedure updates the weights in the input layer of the logic  */
/*  processor.                                                 */
/*_____________________________________________________________*/

void    updateV(double v[][totalInput], double newV[][totalInput])
{
        int     i, j;

        for (i = 0; i < numOfHidden; i++)
        {
                for (j = 0; j < totalInput; j++)
                {
                        v[i][j] = newV[i][j];
                }
        }
}


/*_____________________________________________________________*/
```

```
/*                              updateDeltaW()                                  */
/*______________________________________________________________________________*/
/*      This procedure update the delta values of the weights in the hidden layer of */
/*      the logic processor.                                                    */
/*______________________________________________________________________________*/

void    updateDeltaW(double oldDw[], double newDw[])
{
        int     i;

        for (i = 0; i < numOfHidden; i++)
        {
                oldDw[i] = newDw[i];
        }
}



/*______________________________________________________________________________*/
/*                              updateDeltaV()                                  */
/*______________________________________________________________________________*/
/*      This procedure updates the delta values of the weights in the input layer */
/*      of the logic processor.                                                 */
/*______________________________________________________________________________*/

void    updateDeltaV(double oldDv[][totalInput], double newDv[][totalInput])
{
      int   i, j;

      for (i = 0; i < numOfHidden; i++)
      {
            for (j = 0; j < totalInput; j++)
            {
                  ldDv[i][j] = newDv[i][j];
            }
      }
}



/*______________________________________________________________________________*/
/*                              outputResult()                                  */
/*______________________________________________________________________________*/
/*      This procedure prints all the outputs of the logic processor.           */
/*______________________________________________________________________________*/

void  outputResult(long int k, double pArry[], double w[], double v[][totalInput], double minError,
        double minW[], double minV[][totalInput], double input[][totalInput], double tarOp[],
        double yout[])
{
      prnIndex(pArry, k);       /* print out the performance index of the LP */
```

```
        prtOutput(w, v);            /* print out the final weights in the LP */
        prtOutputW(w, v);
        prnt(input, tarOp, yout);
}


/*_____________________________________________________________*/
/*                          prnt()                             */
/*_____________________________________________________________*/
/*  This procedure prints all the inputs, target outputs, and the actual outputs  */
/*  of the logic processor.                                    */
/*_____________________________________________________________*/

void   prnt(double input[][totalInput], double tarOp[], double yout[])
{
        int     counter1, counter2;

        for (counter1 = 0; counter1 < numOfData; counter1++)
        {
                printf("Input Data Set %3d.\n", counter1);
                for (counter2 = 0; counter2 < totalInput; counter2++)
                {
                        printf("input[%2d][%2d] = %lf.\n", counter1, counter2, input[counter1][counter2]);
                }
                printf("Target Output = %lf.\n", tarOp[counter1]);
                printf("Actual Output = %lf.\n", yout[counter1]);
                printf("\n\n");
                printf("%s\n",stars);
        }
}


/*_____________________________________________________________*/
/*                          prnIndex()                         */
/*_____________________________________________________________*/
/*  This procedure prints out the performance index.           */
/*_____________________________________________________________*/

void   prnIndex(double pArry[], long int k)
{
        long int        i;

        for (i = 0; i < k; i++)
        {
                printf("%d\t%lf\n", i, pArry[i]);
        }
        printf("\n\n\n");
}
```

```
/*_____________________________________________________________*/
/*                          prtOutput()                        */
/*_____________________________________________________________*/
/*   This procedure prints out the weights of the logic processor.  */
/*_____________________________________________________________*/

void  prtOutput(double w[], double v[][totalInput])
{
      int    i, j;

      printf("the values of Vij are :\n");
      for (i = 0; i < numOfHidden; i++)
      {
            for (j = 0; j < totalInput; j++)
            {
                  printf("v[%2d][%2d] = %lf.\n", i, j, v[i][j]);
            }
            printf("%s\n\n", stars);
            printf("\n");
      }

      printf("\nthe values of wi are :\n");
      for (i = 0; i < numOfHidden; i++)
      {
            printf("w[%2d] = %lf.\n", i, w[i]);
      }
      printf("\n");
}


/*_____________________________________________________________*/
/*                          prtMin()                           */
/*_____________________________________________________________*/
/*   This procedure prints out the minimum weights of the logic processor.  */
/*_____________________________________________________________*/

void  prtMin(double minError, double minW[], double minV[][totalInput])
{
      int    i, j;

      printf("The weights of min. performance index:%lf.\n", minError);

      printf("the values of Vij are :\n");
      for (i = 0; i < numOfHidden; i++)
      {
            for (j = 0; j < totalInput; j++)
            {
                  printf("minV[%2d][%2d] = %lf.\n", i, j, minV[i][j]);
```

```
            }
            printf("\n%s\n", stars);
            printf("\n");
        }

        printf("\nthe values of wi are :\n");
        for (i = 0; i < numOfHidden; i++)
            printf("minW[%2d] = %lf.\n", i, minW[i]);
        printf("\n");
}


void   prtMinW(double minW[], double minV[][totalInput])
{
        int     i, j;

        for (i = 0; i < numOfHidden; i++)
        {
            for (j = 0; j < totalInput; j++)
            {
                printf("%lf\n", minV[i][j]);
            }
        }
        for (i = 0; i < numOfHidden; i++)
            printf("%lf\n", minW[i]);
}


void   prtOutputW(double w[], double v[][totalInput])
{
        int     , j;

        for (i = 0; i < numOfHidden; i++)
        {
            for (j = 0; j < totalInput; j++)
            {
                printf("%lf\n", v[i][j]);
            }
        }
        for (i = 0; i < numOfHidden; i++)
        {
            printf("%lf\n", w[i]);
        }
        printf("\n");
}
```

Appendix C

C.1 Data for the Gas Furnace System

The following are the data for the gas furnace system [1]. $x(t)$ is methane feed rate, and is measured in cubic feet per minute. $y(t)$ is the CO_2 concentration, and is measured in the percentage of CO_2 gas in the off gas generated by the gas furnace system.

t	$x(t)$	$y(t)$	t	$x(t)$	$y(t)$
1	-0.109	53.8	23	0.875	53.2
2	0.000	53.6	24	0.891	52.3
3	0.178	53.5	25	0.987	51.6
4	0.339	53.5	26	1.263	51.2
5	0.373	53.4	27	1.775	50.8
6	0.441	53.1	28	1.976	50.5
7	0.461	52.7	29	1.934	50.0
8	0.348	52.4	30	1.866	49.2
9	0.127	52.2	31	1.832	48.4
10	-0.180	52.0	32	1.767	47.9
11	-0.588	52.0	33	1.608	47.6
12	-1.055	52.4	34	1.265	47.5
13	-1.421	53.0	35	0.790	47.5
14	-1.520	54.0	36	0.360	47.6
15	-1.302	54.9	37	0.115	48.1
16	-0.814	56.0	38	0.088	49.0
17	-0.475	56.8	39	0.331	50.0
18	-0.193	56.8	40	0.645	51.1
19	0.088	56.4	41	0.960	51.8
20	0.435	55.7	42	1.409	51.9
21	0.771	55.0	43	2.670	51.7
22	0.866	54.3	44	2.834	51.2

t	$x(t)$	$y(t)$	t	$x(t)$	$y(t)$
45	2.812	50.0	92	0.943	57.0
46	2.483	48.3	93	0.930	56.0
47	1.929	47.0	94	1.006	54.7
48	1.485	45.8	95	1.137	53.2
49	1.214	45.6	96	1.198	52.1
50	1.239	46.0	97	1.054	51.6
51	1.608	46.9	98	0.595	51.0
52	1.905	47.8	99	-0.080	50.5
53	2.023	48.2	100	-0.341	50.4
54	1.815	48.3	101	-0.288	51.0
55	0.535	47.9	102	-0.153	51.8
56	0.122	47.2	103	-0.109	52.4
57	0.009	47.2	104	-0.187	53.0
58	0.164	48.1	105	-0.255	53.4
59	0.671	49.4	106	-0.299	53.6
60	1.019	50.6	107	-0.007	53.7
61	1.146	51.5	108	0.254	53.8
62	1.155	51.6	109	0.330	53.8
63	1.112	51.2	110	0.102	53.8
64	1.121	50.5	111	-0.423	53.3
65	1.223	50.1	112	-1.139	53.0
66	1.257	49.8	113	-2.275	52.9
67	1.157	49.6	114	-2.594	53.4
68	0.913	49.4	115	-2.716	54.6
69	0.620	49.3	116	-2.510	56.4
70	0.255	49.2	117	-1.790	58.0
71	-0.280	49.3	118	-1.346	59.4
72	-1.080	49.7	119	-1.081	60.2
73	-1.551	50.3	120	-0.910	60.0
74	-1.799	51.3	121	-0.876	59.4
75	-1.825	52.8	122	-0.885	58.4
76	-1.456	54.4	123	-0.800	57.6
77	-0.944	56.0	124	-0.544	56.9
78	-0.570	56.9	125	-0.416	56.4
79	-0.431	57.5	126	-0.271	56.0
80	-0.577	57.3	127	0.000	55.7
81	-0.960	56.6	128	0.403	55.3
82	-1.616	56.0	129	0.841	55.0
83	-1.875	55.4	130	1.285	54.4
84	-1.891	55.4	131	1.607	53.7
85	-1.746	56.4	132	1.746	52.8
86	-1.474	57.2	133	1.683	51.6
87	-1.201	58.0	134	1.485	50.6
88	-0.927	58.4	135	0.993	49.4
89	-0.524	58.4	136	0.648	48.8
90	0.040	58.1	137	0.577	48.5
91	0.788	57.7	138	0.577	48.7

t	*x(t)*	*y(t)*	*t*	*x(t)*	*y(t)*
139	0.632	49.2	186	0.084	55.9
140	0.747	49.8	187	0.000	55.9
141	0.900	50.4	188	0.001	55.2
142	0.993	50.7	189	0.209	54.4
143	0.968	50.9	190	0.556	53.7
144	0.790	50.7	191	0.782	53.6
145	0.399	50.5	192	0.858	53.6
146	-0.161	50.4	193	0.918	53.2
147	-0.553	50.2	194	0.862	52.5
148	-0.603	50.4	195	0.416	52.0
149	-0.424	51.2	196	-0.336	51.4
150	-0.194	52.3	197	-0.959	51.0
151	-0.049	53.2	198	-1.813	50.9
152	0.060	53.9	199	-2.378	52.4
153	0.161	54.1	200	-2.499	53.5
154	0.301	54.0	201	-2.473	55.6
155	0.517	53.6	202	-2.330	58.0
156	0.566	53.2	203	-2.053	59.5
157	0.560	53.0	204	-1.739	60.0
158	0.573	52.8	205	-1.261	60.4
159	0.592	52.3	206	-0.569	60.5
160	0.671	51.9	207	-0.137	60.2
161	0.933	51.6	208	-0.024	59.7
162	1.337	51.6	209	-0.050	59.0
163	1.460	51.4	210	-0.135	57.6
164	1.353	51.2	211	-0.276	56.4
165	0.772	50.7	212	-0.534	55.2
166	0.218	50.0	213	-0.871	54.5
167	-0.237	49.4	214	-1.243	54.1
168	-0.714	49.3	215	-1.439	54.1
169	-1.099	49.7	216	-1.422	54.4
170	-1.269	50.6	217	-1.175	55.5
171	-1.175	51.8	218	-0.813	56.2
172	-0.676	53.0	219	-0.634	57.0
173	0.033	54.0	220	-0.582	57.3
174	0.556	55.3	221	-0.625	57.4
175	0.643	55.9	222	-0.713	57.0
176	0.484	55.9	223	-0.848	56.4
177	0.109	54.6	224	-1.039	55.9
178	-0.310	53.5	225	-1.346	55.5
179	-0.697	52.4	226	-1.628	55.3
180	-1.047	52.1	227	-1.619	55.2
181	-1.218	52.3	228	-1.149	55.4
182	-1.183	53.0	229	-0.488	56.0
183	-0.873	53.8	230	-0.160	56.5
184	-0.336	54.6	231	-0.007	57.1
185	0.063	55.4	232	-0.092	57.3

t	*x(t)*	*y(t)*	*t*	*x(t)*	*y(t)*
233	-0.620	56.8	265	1.032	52.0
234	-1.086	55.6	266	0.866	54.0
235	-1.525	55.0	267	0.527	55.1
236	-1.858	54.1	268	0.093	54.5
237	-2.029	54.3	269	-0.458	52.8
238	-2.024	55.3	270	-0.748	51.4
239	-1.961	56.4	271	-0.947	50.8
240	-1.952	57.2	272	-1.029	51.2
241	-1.794	57.8	273	-0.928	52.0
242	-1.302	58.3	274	-0.645	52.8
243	-1.030	58.6	275	-0.424	53.8
244	-0.918	58.8	276	-0.276	54.5
245	-0.798	58.8	277	-0.159	54.9
246	-0.867	58.6	278	-0.033	54.9
247	-1.047	58.0	279	0.102	54.8
248	-1.123	57.4	280	0.251	54.4
249	-0.876	57.0	281	0.280	53.7
250	-0.395	56.4	282	0.000	53.3
251	0.185	56.3	283	-0.493	52.8
252	0.662	56.4	284	-0.759	52.6
253	0.709	56.4	285	-0.824	52.6
254	0.605	56.0	286	-0.740	53.0
255	0.501	55.2	287	-0.528	54.3
256	0.603	54.0	288	-0.204	56.0
257	0.943	53.0	289	0.034	57.0
258	1.223	52.0	290	0.204	58.0
259	1.249	51.6	291	0.253	58.6
260	0.824	51.6	292	0.195	59.5
261	0.102	51.1	293	0.131	58.3
262	0.025	50.4	294	0.017	57.8
263	0.382	50.0	295	-0.182	57.3
264	0.922	50.0	296	-0.262	57.0

Appendix D

The weights in the original LPs and the induced boolean LPs in which for modeling the Box-Jenkins data are shown in the next few pages.

Weights in the LP_{y_low}

	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8
$x\_negative(t-2)$	0.769	0.337	0.420	1.000	0.500	1.000	0.653	0.223
$x\_zero(t-2)$	0.557	0.957	0.505	1.000	0.351	1.000	0.476	0.335
$x\_positive(t-2)$	0.000	0.468	0.867	0.000	0.144	0.700	0.000	0.362
$y\_low(t)$	0.554	0.000	0.320	1.000	0.239	0.000	0.728	0.940
$y\_medium(t)$	0.939	0.821	0.619	1.000	0.541	1.000	0.477	0.152
$y\_high(t)$	1.000	0.180	0.816	1.000	0.349	1.000	0.544	0.543
$y\_low(t-1)$	0.361	0.464	0.339	0.349	0.309	1.000	0.193	0.007
$y\_medium(t-1)$	0.112	0.130	0.172	0.745	0.382	1.000	0.662	0.485
$y\_high(t-1)$	0.394	0.576	0.892	1.000	0.428	1.000	0.722	0.826
$\overline{x\_negative(t-2)}$	0.688	0.251	0.023	0.761	0.140	0.000	0.366	0.744
$\overline{x\_zero(t-2)}$	0.282	0.469	0.815	0.509	0.445	1.000	0.449	0.443
$\overline{x\_positive(t-2)}$	0.900	0.502	0.794	1.000	0.547	1.000	0.656	0.228
$\overline{y\_low(t)}$	0.097	0.614	0.528	1.000	0.050	1.000	0.160	0.427
$\overline{y\_medium(t)}$	0.979	0.763	0.041	1.000	0.810	0.988	0.941	0.890
$\overline{y\_high(t)}$	0.054	0.931	0.087	0.833	0.695	0.384	0.703	0.672
$\overline{y\_low(t-1)}$	0.971	0.936	0.276	0.574	0.564	1.000	0.304	0.393
$\overline{y\_medium(t-1)}$	0.531	0.719	0.959	1.000	0.802	1.000	0.887	0.702
$\overline{y\_high(t-1)}$	0.814	0.138	0.519	0.629	0.400	0.803	0.444	0.630

	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8
$y\_low(t+1)$	0.857	0.588	0.130	1.000	0.239	1.000	0.719	0.013

Weights in the LP_{y_medium}

	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8
$x\_negative(t-2)$	0.768	0.285	0.419	0.723	0.503	1.000	1.000	0.228
$x\_zero(t-2)$	0.509	0.954	0.477	0.123	0.349	1.000	1.000	0.336
$x\_positive(t-2)$	0.131	0.470	0.869	0.509	0.204	1.000	1.000	0.380
$y\_low(t)$	0.611	0.040	0.433	0.725	0.289	1.000	1.000	0.945
$y\_medium(t)$	0.903	0.817	0.587	0.174	0.541	0.262	0.152	0.152
$y\_high(t)$	1.000	0.095	0.823	1.000	0.349	1.000	1.000	0.546
$y\_low(t-1)$	0.368	0.458	0.443	0.171	0.333	1.000	1.000	0.054
$y\_medium(t-1)$	0.122	0.107	0.177	0.558	0.386	1.000	1.000	0.487
$y\_high(t-1)$	0.159	0.551	0.895	0.915	0.428	1.000	1.000	0.827
$\overline{x\_negative(t-2)}$	0.704	0.251	0.041	0.864	0.141	0.984	0.000	0.745
$\overline{x\_zero(t-2)}$	0.297	0.474	0.851	0.578	0.445	1.000	1.000	0.445
$\overline{x\_positive(t-2)}$	0.818	0.498	0.751	0.906	0.547	0.000	1.000	0.227
$\overline{y\_low(t)}$	0.090	0.607	0.469	0.808	0.038	1.000	0.120	0.427
$\overline{y\_medium(t)}$	0.971	0.764	0.164	0.291	0.817	1.000	1.000	0.890
$\overline{y\_high(t)}$	0.059	0.931	0.093	0.842	0.697	0.000	1.000	0.673
$\overline{y\_low(t-1)}$	0.961	0.929	0.269	0.526	0.564	1.000	0.644	0.395
$\overline{y\_medium(t-1)}$	0.488	0.713	0.969	0.378	0.802	0.777	0.828	0.705
$\overline{y\_high(t-1)}$	0.824	0.138	0.551	0.655	0.399	0.918	1.000	0.630

	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8
$y\_medium(t+1)$	0.800	0.565	0.220	0.990	0.260	1.000	1.000	0.063

Weights in the LP_{y_high}

	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8
$x\_negative(t-2)$	0.765	0.259	0.416	0.186	0.468	0.523	1.000	0.177
$x\_zero(t-2)$	0.512	0.964	0.481	0.288	0.348	0.209	1.000	0.355
$x\_positive(t-2)$	0.102	0.482	0.868	0.357	0.167	0.645	1.000	0.389
$y\_low(t)$	0.409	0.160	0.387	0.617	0.217	0.303	1.000	0.947
$y\_medium(t)$	0.891	0.824	0.594	0.303	0.538	0.930	1.000	0.153
$y\_high(t)$	0.889	0.000	0.792	0.799	0.324	0.373	0.081	0.543
$y\_low(t-1)$	0.197	0.522	0.396	0.184	0.277	0.230	1.000	0.128
$y\_medium(t-1)$	0.104	0.137	0.169	0.450	0.378	0.449	1.000	0.484
$y\_high(t-1)$	0.000	0.599	0.853	0.779	0.399	0.391	1.000	0.826
$\overline{x\_negative(t-2)}$	0.726	0.316	0.059	0.945	0.149	0.422	1.000	0.754
$\overline{x\_zero(t-2)}$	0.266	0.469	0.821	0.216	0.445	0.588	0.780	0.440
$\overline{x\_positive(t-2)}$	0.795	0.499	0.748	0.850	0.546	0.000	0.000	0.224
$\overline{y\_low(t)}$	0.091	0.607	0.465	0.646	0.025	0.239	0.120	0.426
$\overline{y\_medium(t)}$	0.946	0.784	0.065	0.001	0.791	0.285	1.000	0.890
$\overline{y\_high(t)}$	0.063	0.936	0.103	0.844	0.698	0.633	1.000	0.676
$\overline{y\_low(t-1)}$	0.958	0.928	0.259	0.514	0.563	0.554	0.115	0.391
$\overline{y\_medium(t-1)}$	0.460	0.741	0.951	0.337	0.799	0.198	1.000	0.707
$\overline{y\_high(t-1)}$	0.818	0.155	0.546	0.657	0.398	0.824	1.000	0.630

	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8
$y\_high(t+1)$	0.685	0.641	0.000	0.875	0.144	0.628	1.000	0.051

Weights in the induced boolean LP_{y_low}

	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8
$x\_negative(t-2)$	1	0	0	1	0	1	0	0
$x\_zero(t-2)$	0	1	0	1	0	1	0	0
$x\_positive(t-2)$	0	0	1	0	0	1	0	0
$y\_low(t)$	0	0	0	1	0	0	1	1
$y\_medium(t)$	1	1	0	1	0	1	0	0
$y\_high(t)$	1	0	1	1	0	1	0	0
$y\_low(t-1)$	0	0	0	0	0	1	0	0
$y\_medium(t-1)$	0	0	0	1	0	1	0	0
$y\_high(t-1)$	0	0	1	1	0	1	1	1
$\overline{x\_negative(t-2)}$	0	0	0	1	0	0	0	1
$\overline{x\_zero(t-2)}$	0	0	1	0	0	1	0	0
$\overline{x\_positive(t-2)}$	1	0	1	1	0	1	0	0
$\overline{y\_low(t)}$	0	0	0	1	0	1	0	0
$\overline{y\_medium(t)}$	1	1	0	1	1	1	1	1
$\overline{y\_high(t)}$	0	1	0	1	0	0	1	0
$\overline{y\_low(t-1)}$	1	1	0	0	0	1	0	0
$\overline{y\_medium(t-1)}$	0	1	1	1	1	1	1	1
$\overline{y\_high(t-1)}$	1	0	0	0	0	1	0	0

	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8
$y\_low(t+1)$	0	0	0	1	0	1	0	0

Weights in the induced boolean LP_{y_medium}

	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8
$x\_negative(t-2)$	1	0	0	1	0	1	1	0
$x\_zero(t-2)$	0	1	0	0	0	1	1	0
$x\_positive(t-2)$	0	0	1	0	0	1	1	0
$y\_low(t)$	1	0	0	1	0	1	1	1
$y\_medium(t)$	1	1	0	0	0	0	0	0
$y\_high(t)$	1	0	1	1	0	1	1	0
$y\_low(t-1)$	0	0	0	0	0	1	1	0
$y\_medium(t-1)$	0	0	0	0	0	1	1	0
$y\_high(t-1)$	0	0	1	1	0	1	1	1
$\overline{x\_negative(t-2)}$	1	0	0	1	0	1	0	1
$\overline{x\_zero(t-2)}$	0	0	1	0	0	1	1	0
$\overline{x\_positive(t-2)}$	1	0	1	1	0	0	1	0
$\overline{y\_low(t)}$	0	1	0	1	0	1	0	0
$\overline{y\_medium(t)}$	1	1	0	0	1	1	1	1
$\overline{y\_high(t)}$	0	1	0	1	1	0	1	1
$\overline{y\_low(t-1)}$	1	1	0	0	0	1	1	0
$\overline{y\_medium(t-1)}$	0	1	1	0	1	1	1	1
$\overline{y\_high(t-1)}$	1	0	0	1	0	1	1	1

	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8
$y\_medium(t+1)$	0	0	0	1	0	1	1	0

Weights in the induced boolean LP_{y_high}

	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8
$x\_negative(t-2)$	1	0	0	0	0	0	1	0
$x\_zero(t-2)$	0	1	0	0	0	0	1	0
$x\_positive(t-2)$	0	0	1	0	0	0	1	0
$y\_low(t)$	0	0	0	0	0	0	1	1
$y\_medium(t)$	1	1	0	0	0	1	1	0
$y\_high(t)$	1	0	1	1	0	0	0	0
$y\_low(t-1)$	0	0	0	0	0	0	1	0
$y\_medium(t-1)$	0	0	0	0	0	0	1	0
$y\_high(t-1)$	0	0	1	1	0	0	1	1
$\overline{x\_negative(t-2)}$	1	0	0	1	0	0	1	1
$\overline{x\_zero(t-2)}$	0	0	1	0	0	0	1	0
$\overline{x\_positive(t-2)}$	1	0	1	1	0	0	0	0
$\overline{y\_low(t)}$	0	0	0	0	0	0	0	0
$\overline{y\_medium(t)}$	1	1	0	0	1	0	1	1
$\overline{y\_high(t)}$	0	1	0	1	0	0	1	0
$\overline{y\_low(t-1)}$	1	1	0	0	0	0	0	0
$\overline{y\_medium(t-1)}$	0	1	1	0	1	0	1	1
$\overline{y\_high(t-1)}$	1	0	0	0	0	1	1	0

	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8
$y\_high(t+1)$	0	0	0	0	0	0	1	0