PRECISIONSM RESOLUTION TARGETS

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file - Votre référence

Our file - Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

A Sparse Optimization Approach to MLP Training and Its Applications

by

Fang Wang, B. Eng.,

A thesis submitted to the Faculty of Graduate Studies and Research

in partial fulfillment of the requirements for the degree of

Master of Engineering

Ottawa-Carleton Institute for Electrical Engineering

Department of Electronics

Carleton University

Ottawa, Ontario K1S 5B6

Canada

©Copyright August 1995, Fang Wang

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file / Votre référence

Our file / Notre référence

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-612-08968-1

Canada

Name

Dany. Furg

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.

1. Education Research - Teacher Training

"SUBJECT TERM"

0544

SUBJECT CODE

U·M·I

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

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

EDUCATION

General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523

Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language 0679
General 0289
Ancient 0290
Linguistics 0291
Modern 0291
Literature 0401
General 0294
Classical 0295
Comparative 0297
Medieval 0298
Modern 0316
African 0591
American 0305
Asian 0352
Canadian (English) 0355
Canadian (French) 0593
English 0311
Germanic 0312
Latin American 0315
Middle Eastern 0313
Romance 0314
Slavic and East European 0314

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy 0422
Religion 0318
General 0321
Biblical Studies 0319
Clergy 0320
History of 0322
Philosophy of 0469
Theology 0323

SOCIAL SCIENCES

American Studies 0323
Anthropology 0324
Archaeology 0326
Cultural 0327
Physical 0310
Business Administration 0272
General 0770
Accounting 0454
Banking 0338
Management 0385
Marketing 0501
Canadian Studies 0503
Economics 0505
General 0508
Agricultural 0509
Commerce Business 0510
Finance 0511
History 0358
Labor 0366
Theory 0351
Folklore 0578
Geography 0366
Gerontology 0351
History 0578
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 0615
General 0616
International Law and Relations 0617
Public Administration 0814
Recreation 0452
Social Work 0626
Sociology 0627
General 0938
Criminology and Penology 0631
Demography 0628
Ethnic and Racial Studies 0629
Individual and Family Studies 0630
Industrial and Labor Relations 0700
Public and Social Welfare 0709
Social Structure and Development 0999
Theory and Methods 0453
Transportation 0453
Urban and Regional Planning 0453
Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture 0473
General 0285
Agronomy 0475
Animal Culture and Nutrition 0476
Animal Pathology 0359
Food Science and Technology 0478
Forestry and Wildlife 0479
Plant Culture 0480
Plant Pathology 0817
Plant Physiology 0777
Range Management 0746
Wood Technology 0306
Biology 0287
General 0308
Anatomy 0309
Biostatistics 0379
Botany 0329
Cell 0353
Ecology 0369
Entomology 0793
Genetics 0410
Immunology 0307
Microbiology 0317
Molecular 0416
Neuroscience 0433
Oceanography 0821
Physiology 0778
Radiation 0472
Veterinary Science 0786
Zoology 0760
Biophysics 0786
General 0760
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 0566
General 0300
Audiology 0992
Chemotherapy 0567
Dentistry 0350
Education 0769
Hospital Management 0758
Human Development 0982
Immunology 0564
Medicine and Surgery 0347
Mental Health 0569
Nursing 0570
Nutrition 0380
Obstetrics and Gynecology 0354
Occupational Health and Therapy 0381
Ophthalmology 0571
Pathology 0719
Pharmacology 0572
Pharmacy 0573
Physical Therapy 0574
Public Health 0575
Radiology 0575
Recreation 0575

Speech Pathology 0460
Toxicology 0383
Home Economics 0386

PHYSICAL SCIENCES

Pure Sciences 0485
Chemistry 0749
General 0486
Agricultural 0487
Analytical 0488
Biochemistry 0738
Inorganic 0490
Nuclear 0491
Organic 0494
Pharmaceutical 0495
Physical 0754
Polymer 0405
Radiation 0605
Mathematics 0986
Physics 0606
General 0608
Acoustics 0748
Astronomy and Astrophysics 0507
Atmospheric Science 0798
Atomic 0759
Electronics and Electricity 0609
Elementary Particles and High Energy 0610
Fluid and Plasma 0752
Molecular 0756
Nuclear 0611
Optics 0463
Radiation 0346
Solid State 0984
Statistics 0984

Applied Sciences

Applied Mechanics 0346
Computer Science 0984

Engineering 0453
General 0539
Aerospace 0540
Agricultural 0541
Automotive 0542
Biomedical 0543
Chemical 0544
Civil 0348
Electronics and Electrical 0545
Heat and Thermodynamics 0546
Hydraulic 0547
Industrial 0794
Marine 0548
Materials Science 0743
Mechanical 0551
Metallurgy 0552
Mining 0549
Nuclear 0765
Packaging 0554
Petroleum 0790
Sanitary and Municipal System Science 0428
Geotechnology 0796
Operations Research 0795
Plastics Technology 0994
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

The undersigned recommend
the Faculty of Graduate Studies and Research
acceptance of the thesis

A Sparse Optimization Approach to MLP Training and Its Applications

submitted by Fang Wang, B. Eng.,
in partial fulfillment of the requirements for
the degree of Master of Engineering

Q.J. Zhang, Thesis Supervisor

J. Wight, Chair, Department of Electronics

Abstract

This work was motivated by the potential of neural networks in electrical design applications and the increasing need for a fast and efficient training algorithm to ensure the success of such expanding applications. A new and efficient sparse backpropagation training approach is proposed in this thesis. The approach is formulated by combining the sparse optimization concept, substantially used in circuit optimization areas, with the backpropagation method, which is an existing and influential training technique for Multilayer Perceptrons (MLP). The sparse phenomenon due to neuron activations, which is inherent in neural networks, is exploited. A new set of backpropagation formulae with embedded sparse optimization concept is derived. A systematic computational algorithm is developed and tested that implements the sparse training approach in an efficient way. A completely new set of parameters that regulate the sparse learning process is introduced and an adaptation procedure for the new training parameters is proposed.

The validity and efficiency of the proposed training algorithm are confirmed by two different types of applications, i.e., speech recognition and circuit extraction. Speech recognition is an existing and challenging area for neural network applications. The proposed sparse training algorithm is applied to the training of MLP using the Texas Instruments isolated speech data TI-20 and TI-46. Excellent recognition accuracy is achieved while the proposed sparse training provided 2 to 10 times training speedup over standard backpropagation.

The second type of application is a new application area for neural networks, i.e., circuit extraction. A new formulation based on multilayer perceptrons is developed to

learn the circuit information and to use the knowledge to extract the circuit macro-models. Again the application of the proposed sparse training technique provided significant speedup over standard training approaches.

Acknowledgments

Firstly, my grateful appreciation is extended to my supervisor Prof. Qi-jun Zhang for his encouragement, continued assistance, expert guidance and supervision throughout the course of my graduate studies and research. My great thanks also go to Prof. Michel Nakhla for his invaluable advice and comments on the research work.

I wish to thank all the staff and fellow graduate students in the Department of Electronics for their assistance and friendship. I am indebted to them for occupying so much of computing hours, sometimes even slowing down the network. A special thank you to all the graduate students in the CAD group for the cheer and friendship making my study an enjoyable experience.

Finally, I wish to take this opportunity to express my deepest appreciation to my parents and my sister. Without their years of encouragement and endless support, I would not have the opportunity to complete this work.

Contents

1	Introduction	1
1.1	Motivation of the Research Work	1
1.2	Thesis Objective	3
1.3	Outline of the Thesis	4
2	Literature Review	7
2.1	Recent Development in Neural Network Applications to Circuit Problems	7
2.2	Review of Training Methods for Multilayer Perceptrons	9
2.3	Sparse Optimization	14
3	A Sparse Optimization Approach to MLP Training	17
3.1	Standard Backpropagation: Dense Training	18
3.2	Sparse Concept in MLP Neural Network Training	20
3.3	Sparse Backpropagation in General Multilayer Perceptrons	25
3.4	Algorithms for Sparse Backpropagation Training of General MLP . .	29
3.4.1	Algorithm 1 (SEBP1)	29
3.4.2	Algorithm 2 (SEBP2)	32
3.5	Stop Criteria	35
3.6	Random Perturbation	35
3.7	Sparse Training Parameters and Their Adaptation	36

3.7.1	Learning Rate Adaptation	37
3.7.2	Sparse Threshold	38
3.7.3	Sparse Pattern Refreshment Interval	40
3.8	Conclusion	40
4	Application to Speech Recognition	43
4.1	Introduction	44
4.2	Speech Recognition System	46
4.3	Speech Signal Preprocessing	47
4.4	Clustering Algorithm for Codebook Development	47
4.4.1	Introduction	48
4.4.2	Algorithm	49
4.4.3	Examples	51
4.5	The MLP Neural Network and Its Training	53
4.6	Examples by Sparse Training Approach	56
4.6.1	Example 1	56
4.6.2	Example 2	58
4.6.3	Example 3	60
4.7	Conclusion	61
5	Application to Circuit Extraction	75
5.1	Introduction	75
5.2	Problem Description	77
5.3	MLP Formulation for Circuit Extraction	78
5.4	An Example of MLP Inputs and Outputs	84
5.5	Neural Network Training by the Sparse Approach	89

5.5.1	Example 1	89
5.5.2	Example 2	91
5.6	Conclusion	93
6	Conclusions and Further Research	102
6.1	Conclusions	102
6.2	Suggestions for Future Research	104
A	Instructions for Using the Sparse MLP Program	118

List of Figures

3.1	Architecture of the neural network.	18
3.2	Feedforward path in (a) dense training approach, and (b)- (c) sparse training approach. The feedforward path in the sparse training approach is different depending on the training data as shown in (b) for training pattern 1, and (c) for training pattern 2.	22
3.3	Backpropagation path in (a) dense training approach, and (b)-(c) sparse training approach. The backpropagation path in the sparse training approach is different depending on the training data as shown in (b) for training pattern 1, and (c) for training pattern 2.	23
3.4	General Multilayer Perceptrons.	26
3.5	Flow diagram of sparse-concept based training Algorithm 2.	42
4.1	Diagram of combined multi-codebook/MLP neural network speech recognition system.	63
4.2	Distribution of sparse factor ρ_0^1 for training patterns in the speaker dependent case with 20 words vocabulary.	64
4.3	Learning curves of EBP algorithm('-.'), SEBP1 algorithm('-.') and SEBP2 algorithm('-.') for the speaker dependent case with 20 words vocabulary.	65

4.4	Distribution of sparse factors ρ_0^1 for training patterns in the multi-speaker dependent case with 20 words vocabulary.	66
4.5	Learning curves of EBP algorithm('-.'), SEBP1 algorithm(' - ') and SEBP2 algorithm(' - ') for the multi-speaker dependent case with 20 words vocabulary (2 layer network).	67
4.6	Learning curves of EBP algorithm('-.'), SEBP1 algorithm(' - ') and SEBP2 algorithm(' - ') for the multi-speaker dependent case with 20 words vocabulary (3 layer network).	68
4.7	Histogram of sparse factor ρ_1^2 before training for multi-speaker dependent case with 20 words vocabulary.	69
4.8	Histogram of sparse factor ρ_1^2 after training for multi-speaker dependent case with 20 words vocabulary.	69
4.9	Histogram of sparse factor ρ_0^2 before training for multi-speaker dependent case with 20 words vocabulary.	70
4.10	Histogram of sparse factor ρ_0^2 after training for multi-speaker dependent case with 20 words vocabulary.	70
4.11	Distribution of sparse factors ρ_0^1 for training patterns in the multi-speaker dependent case with 46 words vocabulary.	71
4.12	Learning curves of EBP algorithm('-.'), SEBP1 algorithm(' - ') and SEBP2 algorithm(' - ') for the multi-speaker dependent case with 46 words vocabulary.	72
4.13	Histogram of sparse factor ρ_1^2 before training for multi-speaker dependent case with 46 words vocabulary.	73
4.14	Histogram of sparse factor ρ_1^2 after training for multi-speaker dependent case with 46 words vocabulary.	73

4.15	Histogram of sparse factor ρ_0^2 before training for multi-speaker dependent case with 46 words vocabulary.	74
4.16	Histogram of sparse factor ρ_0^2 after training for multi-speaker dependent case with 46 words vocabulary.	74
5.1	The format of the connection matrix C and component type matrix T for the extraction problem. These two matrices form the input feature vector of the MLP. Each numerical entry in the matrices is either 1 or 0. The values in this figure are for illustration purpose only.	81
5.2	The format of outputs of the MLP circuit extractor. Each numerical entry in D and H is either 1 or 0. The values in this figure are for illustration purpose only.	83
5.3	MLP input parameters for the gate NOT (inverter).	87
5.4	MLP output parameters for the gate NOT (inverter).	88
5.5	Learning curves of EBP algorithm('-.') and SEBP2 algorithm('-.') for NOT and TGATE extraction in the full model case.	95
5.6	Learning curves of EBP algorithm('-.') and SEBP2 algorithm('-.') for NOT and TGATE extraction in the partial model case.	96
5.7	Distribution of sparse factor ρ_0^1 for training patterns for NAND-NOR extraction in the partial model case.	97
5.8	Histogram of input layer sparse factor ρ_0^1 for NAND-NOR extraction in the partial model case.	97
5.9	Learning curves of EBP algorithm('-.') and SEBP2 algorithm('-.') for NAND and NOR extraction in the full model case.	98

5.10	Learning curves of EBP algorithm('-.') and SEBP2 algorithm('-.') for NAND and NOR extraction in the partial model case.	99
5.11	Histogram of sparse factor ρ_1^2 before training for NAND-NOR extrac- tion in the partial model case.	100
5.12	Histogram of sparse factor ρ_1^2 after training for NAND-NOR extraction in the partial model case.	100
5.13	Histogram of sparse factor ρ_0^2 before training for NAND-NOR extrac- tion in the partial model case.	101
5.14	Histogram of sparse factor ρ_0^2 after training for NAND-NOR extraction in the partial model case.	101

List of Tables

4.1	Comparison Between Binary Splitting Algorithm and the Nonbinary Algorithm for Cepstral Codebook	52
4.2	Comparison Between Binary Splitting Algorithm and the Nonbinary Algorithm for Delta-cepstral Codebook	52
4.3	Comparison Between Binary Splitting Algorithm and the Nonbinary Algorithm for Delta-delta-cepstral Codebook	52
4.4	Comparison Between Binary Splitting Algorithm and the Nonbinary Algorithm for Energy Codebook	53
4.5	Training and Recognition Results for the TI 20 Data (speaker dependent)	58
4.6	Training and Recognition Results for the TI 20 Data (multi-speaker dependent)	60
4.7	Training and Recognition Results for the TI 46 Data (multi-speaker dependent)	61
5.1	NOT-TGATE Case: List of Number of MLP Inputs, Outputs and Patterns	90
5.2	NOT-TGATE Case: Comparison of EBP and Sparse Training Approach SEBP2	91
5.3	NAND-NOR Case: List of Number of MLP Inputs, Outputs and Patterns	92

5.4 NAND-NOR Case: Comparison of EBP and Sparse Training Approach

SEBP2	93
------------------------	-----------

Chapter 1

Introduction

1.1 Motivation of the Research Work

The rapid development of today's electronics and telecommunications industry resulted in many circuit design and analysis problems which are large, complicated and computationally intensive. There is a recent thrust of research in using neural network techniques to solve some of the difficult problems in electronics design. Neural networks enjoy the advantage to learn from given examples, and the ability to generalize. Once trained by corresponding electrical data, neural networks have been used for rapid estimation of circuit and device performances in highly repetitive analysis problems such as circuit optimization and statistical design.

However one of the serious concerns raised from such neural network applications, is the extremely long training time required by neural network learning. Many circuit problems are highly nonlinear, and circuit design parameter space is of high dimension. Consequently the size and accuracy requirements of neural networks for this application further make training very time consuming. In cases where a library of models, e.g., transistor models, or VLSI interconnect models, are needed to compose an overall simulation problem, the training time required for library development be-

comes very long. Also to update the library with new components requires additional training. How to speedup the training process is not only an interesting topic by itself, but also a crucial step for the success in its new applications in the electronic design area.

To improve learning speed of neural networks, of course, has been an important topic and investigated intensively by many researchers in the past. Many advanced algorithms have been proposed, such as advanced adaptation mechanisms for training parameters and/or neural network structures. Neural network training is an optimization process. Several types of optimization concepts, such as stochastic optimization, conjugate gradient based optimization, and quasi-Newton algorithms have been used. However compared to standard optimization examples, neural network training has its particular challenges since it involves a large number of weightings as optimization variables. The existence of activation functions such as sigmoid functions, whose derivatives are virtually zero in much of its parameter space, further complicates the convergence of optimization.

On the other hand, electrical optimizations have also faced similar types of challenges, i.e., large-scale optimization involving many electrical variables and design specifications, and difficulties arising from sensitivity information. Addressing these difficulties has been an important area for many years in circuit design and several types of advanced optimization approaches have been developed. One of the advanced approaches, which is based on sparse matrix and decomposition concept, has been particularly effective in solving large-scale engineering optimization problems. Is it possible to combine this concept with neural network training process, and to exploit new potentials in speeding up neural network training? Addressing this technical gap is one of the motivations in this thesis.

The neural network approach in circuit applications is still in its early stage. Continued research in this direction is currently underway. As the technique becomes more suited to electrical applications, including faster training and new structures of neural network development, more electrical design problems can be alternatively formulated into neural network solutions, and perhaps be better solved using the neural network approach. To continue exploring the potential of neural network concept for circuit applications and to bridge the gap between conventional electrical approaches and neural network approaches in circuit problems is another motivation of this thesis work.

1.2 Thesis Objective

The objective of this thesis is to develop new algorithms to bridge several gaps between conventional neural network techniques and electrical design areas. The first goal is to combine a sparse optimization concept, substantially used in circuit design, with the backpropagation method, which is the most influential training technique for multilayer perceptrons. Clearly it is needed to investigate if there is any sparsity in the neural networks. This mathematical concept while much easier to identify in electrical problems, however, is not very straightforward in a neural network environment. A more careful examination and a new view of the neural network mechanism are needed. Another challenge will be how to formulate the training problem into a suitable form so that the sparse concept can be applied. The ultimate goal is to develop a new algorithm to further improve neural network training efficiency by taking advantage of the sparse concept whose potential has not been fully exploited in conventional training algorithms.

To validate the proposed training algorithm, we will use two completely different types of application examples. The first application is speech recognition, which is chosen because it is one of the challenge areas for neural networks already under intensive investigation. The success of the new training algorithm in this existing application will be useful to confirm the validity of the proposed approach as a viable alternative to conventional neural network training.

Another objective of the thesis lies in the second type of applications which is from the electrical design and analysis area. We will develop a new approach to an electrical circuit extraction problem, conventionally done using electrical rule-based approaches. Electrical topological information in the original problem needs to be formulated into numerical representations so that it becomes neural network input/output parameters. The subsequent task is to investigate how to use multilayer perceptrons to learn the circuit topological information, and to use this knowledge to extract the circuit macro-models through pattern recognition. The validity of this new formulation needs to be confirmed by extraction examples. Finally the effectiveness of the proposed sparse training technique needs to be confirmed by the training and testing of such circuit extraction examples.

1.3 Outline of the Thesis

This thesis is organized as follows.

In Chapter 2, a review of the literature is presented. This includes a brief review of the neural network technique and its engineering applications with emphasis on recent progress in applying neural networks in circuit analysis and design. A review of the backpropagation training for multilayer perceptrons (MLP) is given, and a

review of the literature in the development of various advanced training techniques is conducted. The concept of sparse matrix approach for numerical optimization and for circuit optimization applications is also reviewed.

Chapter 3 introduces a new and efficient sparse backpropagation approach. This approach is formulated by combining the sparse optimization concept from circuit optimization areas, with the backpropagation training technique for multilayer perceptrons. The sparse phenomenon in neural network is identified from a numerical sparsity point of view. A new set of formulae for both feedforward and backpropagation computations is derived with embedded sparse optimization concept. A systematic computational algorithm is developed and tested that implements the sparse training approach in an efficient way. A completely new set of parameters that regulate the sparse learning process is introduced. An adaptation procedure for all the new training parameters is proposed.

In Chapter 4, the first type of applications for the sparse training technique, i.e., speech recognition, is presented. The background of speech recognition and the description of the speech recognition system used are given. The speech signal preprocessing is presented. A clustering algorithm to achieve better partition result of the feature space is then derived and used for codebook development. Recognition results for the Texas Instruments isolated speech data TI-20 and TI-46 are presented. The sparse phenomenon in the speech recognition MLP system is described. Finally, the proposed sparse-concept based training algorithm is compared with the standard EBP (Error Back Propagation) in the training of the speech recognition system. Much CPU speedup is achieved using the proposed sparse training technique.

Chapter 5 describes the second type of applications of the sparse training technique, i.e., circuit extraction, which is a new application area for neural networks

proposed in this thesis. A new formulation to represent the electrical topological extraction problem by numerical pattern recognition is derived. Neural network input/output parameters are defined. Multilayer perceptrons are experimented with to learn the circuit information and to use this knowledge to extract the circuit macromodels through pattern recognition. The sparse features in this neural network problem and the experiments leading to sparse factors for various testing examples are presented. The efficiency of using the proposed sparse training technique in training the resulting MLP is demonstrated.

Finally, in Chapter 6, conclusions and future suggestions are described.

Chapter 2

Literature Review

2.1 Recent Development in Neural Network Applications to Circuit Problems

To solve practical real-world problems, large scale and highly efficient computational systems are required. The rapid development of today's electronics and telecommunications technology resulted in many circuit design and analysis problems which are large, complicated, and computationally intensive. For example, modeling and optimization are among the most important steps in a circuit design procedure [1] [2] [3] [4], but due to the complexity of the analysis models needed in today's circuit design, optimization with highly repetitive circuit analysis is very computationally demanding [5]. For yield optimization where a large number of circuit analysis is required, the computational complexity is even higher [6]. On the other hand, human brain, which has taken millions of years to evolve to its present architecture, excels at such tasks as vision, speech, information retrieval, and complex spatial and temporal pattern recognition, all of which are extremely difficult tasks for conventional computers. This leads to the study of human brain and thereafter the bloom of artificial neural networks research [7]. Many kinds of architectures

and training algorithms have been developed, such as multilayer perceptrons (MLP), self-organizing maps (SOM), recurrent neural networks, etc. Many applications of neural networks have been developed such as image and speech processing, control and optimization [8] [9]. Neural networks are introduced as trainable devices which have the unique ability to generalize. This appealing feature makes it very attractive to use neural network techniques as an alternative to solve some of the difficult modeling problems in circuit design. Recently there has been a thrust of research in the application of neural network approaches in circuit analysis and optimization.

Neural networks were used in yield optimization of circuits [10] [11], in microwave impedance matching [12], in studying the effects of design factors on printed circuit board (PCB) assembly yield [13], in microwave device modeling, circuit analysis and optimization [14], and in modeling the interconnect delay of high-speed VLSI networks [15] [16]. In these approaches, multilayer perceptrons (MLP) are often used to represent the complicated nonlinear relationships in electrical circuits. MLP can be used with discrete neural net output representations which indicates whether a circuit pass or fail design specifications [10], or used with continuous neural net output which represent the continuous circuit responses [17]. The input variables to the MLP can be continuous variables representing general continuous component values [18] or discrete variables which take only the standard component values adopted in industry [15]. The neural network models can be used in repetitive circuit analysis and optimization to replace the original and slow circuit simulations [18].

2.2 Review of Training Methods for Multilayer Perceptrons

Among various neural network architectures, Multilayer Perceptron (MLP) is one of the most important and most widely used neural networks. They have been applied successfully to solve some difficult and diverse problems by training them in a supervised manner with a highly popular algorithm known as the Error Back-Propagation algorithm (EBP) [19]. This thesis is focussed on this type of neural networks, i.e., on the development of training algorithms and applications of multilayer perceptrons.

The EBP algorithm was proposed by Rumelhart, Hinton and Williams [19]. The basic objective of MLP training is to iteratively adjust the MLP weights such that the MLP outputs match their corresponding target values. The EBP algorithm is based on the steepest descent principle [20] where the weights of the MLP are updated along the negative gradient direction in the weight space. The step size of the updates is controlled by a factor called learning rate.

As neural network research has moved from the state-of-art paradigms to real-world applications, the associated training time and computing requirements have become an increasingly significant consideration [8] [21] [22]. Some of the real world applications involve large-scale systems, in which case the development of fast and efficient learning algorithms becomes extremely important [21]. This has been a subject of considerable interest over the past few years. A variety of techniques have been developed, and among them are two important classes of methods, one being based on advanced learning rate and momentum adaptation, and heuristic rules, and the other being based on the use of advanced optimization techniques.

The basic backpropagation, derived from the principles of steepest descent, is

known to suffer from slow convergence, and possible weight oscillation. The addition of a momentum term to weight update formula as proposed by [19], provided significant improvements to the basic backpropagation reducing weight oscillation. This technique is also known as the generalized delta-rule [9]. Other approaches in reducing oscillation have also been studied, such as invoking a correction term that uses the difference of gradients [23], and the constrained optimization approach where constraints on weights are imposed to achieve better alignment between weight updates in different epoches [24].

An important way to improve the basic backpropagation is to use adaptation schemes to allow the learning rate and the momentum factor to be adaptive during learning [9], e.g., adaptation according to training errors [25]. Many researchers have investigated advanced adaptation schemes to improve the efficiency of training. One of the most interesting work in this area is the delta-bar-delta rule proposed by Jacobs [26]. He developed an algorithm based on a set of heuristics in which the learning rate for different weights are defined separately and also adapted separately during the learning process. The adaptation is determined from two factors, one being the current derivative of the training error with respect to the weights, and the other being an exponentially weighted sum of the current and past derivatives of the training error. Various other adaptation techniques have also been proposed, for example a scheme in which the learning rate was adapted in order to reduce the energy value of the gradient direction in a close-to-optimal way [27], an Enhanced Back Propagation algorithm [28] with a scheme to adapt the learning rate according to values of weights in the neural net, and a learning algorithm inspired from the principle of "forced dynamics" for the total error functional [29]. The algorithm in [29] updates the weights in the direction of steepest descent, but with the learning

rate a specific function of the error and of the error gradient form. An interesting adaptation scheme based on the concept of dynamic learning rate optimization is presented in [30], in which the first and second derivatives of the objective function w.r.t. learning rate from the information gathered during the forward and backward propagation are used. Another work [23], which is considered as an extension of Jocab's heuristics, is to correct the value of the weights near the bottom of a ravine with a new acceleration algorithm based on that correction. This correction term uses the difference of gradients to reduce the oscillation of weights during training. As a general concept in MLP training, the weights are updated iteratively and in each iteration they are updated by a certain step size along an updating direction. The standard backpropagation uses learning rates to adjust the step size, with the advantage that the method is very simple and does not require new computation of the error functions. A different way to determine the step size, is to use line search methods [20] so that the training error is reduced or optimized along the given updating direction. Examples in this category are such as line search based on quadratic models [31], and based on linear interpolation [32] [33]. Other ways to improve training efficiency have also been proposed, such as the gradient reuse algorithm [34]. The basic idea of this method is that gradients which are computed during training are reused until the resulting weight updates no longer lead to a reduction in the error.

Supervised learning for MLP can be viewed as a function optimization problem, where the optimization variables are the MLP weights and the objective functions are the total errors between MLP outputs and their target values. Therefore various optimization concepts can be used, as an alternative to steepest descent which is the base for backpropagation and which is only the simplest of various optimization techniques.

The development of alternative learning algorithms for MLP was attempted by modifying the optimization strategy and/or employing adaptation rules other than the gradient descent, or by considering alternative criteria to the frequently used quadratic error criterion [8]. An important type of training methods is the use of optimization algorithm employing second order derivative information, e.g., [35] [36] [37] [38]. Some of the early work in this area was demonstrated in [35] [36] with the development of second-order learning algorithms for neural networks, and various methods for approximating the second-order derivatives of the objective function iteratively. Examples of continuing developments in this direction are, e.g., a method to approximate the Hessian matrix as diagonal [37], an analytically complicated algorithm using adaptive least squares [38], an Enhanced Training algorithm to further exploit second order information [39] with the recognition that the multilayer perceptron training problem is fundamentally a nonlinear least squares optimization problem. One of the recent investigations in the area is [40] in which exact and efficient methods for calculating second-order derivatives and a range of approximations were discussed. The use of exact second-order information in optimization is, e.g., by the Newton's Method. In reality the second-order derivatives are often approximated from the information of function and its first derivatives, necessitating the use of the quasi-Newton methods in neural network training, e.g., [8] [41], [36]. Another type of optimization methods proposed for neural network training is the conjugate gradient method, e.g., [42], [41]. It has also been proved that the conjugate gradient method is a special case of error backpropagation with momentum term [32]. An interesting review paper is [32] in which several minimization techniques including first order and second order methods such as steepest descent method, Quasi-Newton methods and conjugate gradient methods are reviewed. Most recently, a new learning procedure is presented in [43]

which is based on a linearization of the nonlinear processing elements and the optimization of the multilayer perceptron layer by layer. In order to limit the introduced linearization error a penalty term is added to the cost function. Another alternative to the Error Back Propagation algorithm was based on the combination of gradient descent based minimization and linear programming [44]. This algorithm avoids a wrong direction in the weight space by solving a linear program at each adaptation cycle.

In the feedforward network, slow convergence due to pathological gradients tends to occur in two cases [23]. In one case, the error surface forms a ravine, with the direction of the gradient close to the perpendicular direction of ravine, possibly making the weights oscillate. In the second case, the error surface forms a plane with a gentle slope, and the resulting gradient is too small for weight changes to move rapidly over the wide flat floor (stagnation). The first case is usually solved by using momentum term [19] or other techniques which have the effect of momentum acceleration such as [24]. For the second case, an important method is to use random optimization techniques which are oriented to finding the global minimum. Examples in this class are, e.g., simulated annealing [45] which allows the optimization to jump out of a local minimum through an annealing process controlled by a parameter called temperature [45], the Langevin updating (LV) rule in multilayer perceptron, in which noise is added to the weights during training [46], and a stochastic minimization algorithm for training neural nets [47], which is basically a random optimization method with no gradient information needed. A more general method is the hybrid method which combines the conventional training with random optimization, e.g., [31] which introduced a hybrid algorithm combining the modified backpropagation method (using conjugate gradient methods and line search) and the random optimization method to

find the global minimum of the error function. During training with modified backpropagation method, if a flat error surface is encountered, the training algorithm is switched to the random optimization method. After training escapes from the flat error surface, modified backpropagation method is then switched back [31].

Neural network training has also been viewed from other points of view, e.g., from system identification point of view [48], and from probability interpretation of target activities [49]. To train a neural net and to improve generalization performance, pruning algorithm have been proposed [50] [51], in which case we start with a large multilayer perceptron with an adequate performance for the problem at hand, and then prune it by weakening or eliminating certain synaptic weights in a selective and orderly fashion.

2.3 Sparse Optimization

In the circuit analysis and optimization area, the growing size of circuits and of the optimization problems have also presented serious challenges to researchers and engineers. As the number of circuit components increases, the size of the network equations increases. Standard solution techniques such as LU factorization for linear frequency domain analysis, become very inefficient. For optimization problems, with the increasing number of circuit parameters to be optimized and the increasing number of design specifications at many circuit ports to be satisfied, the problem becomes a large-scale optimization problem. In this case, the conventional optimization methods, while suitable for problems of ordinary size, balk at large problems.

One of the historically very significant advances in solving large circuit problems

is the use of the sparse matrix concept [52], [53]. A matrix is sparse if many of its entries are zero. Conventional dense matrix techniques such as LU factorization involves the computation of all entries in the matrix, and for sparse matrices, much of the computations will involve the unnecessary operations of zeros, reducing computational efficiency. On the other hand, by working with the patterns of zeros in the matrix, the sparse matrix approach explicitly avoids the unnecessary operations of zeros and therefore obtaining substantial CPU speedup while retaining the same numerical solutions as dense techniques. For circuit analysis, as the size of the circuit grows, there is a tendency that many entries in the network matrix are zero, therefore the matrix becomes very sparse [54]. The sparse LU factorization methods can be used in this case. In fact, it is virtually a standard that major commercial circuit simulators use sparse instead of dense matrix solvers e.g., [55]. Standard library routines for sparse LU factorization are also available, e.g., [56].

The sparse matrix concept has also been expanded into optimization. For linear programming techniques, sparse matrix concept has been used and implemented into commercial software such as [57].

For many nonlinear optimization problems it is necessary to estimate the Jacobian of a nonlinear function. Each entry in this Jacobian matrix represents the sensitivity between a function and a variable. In large scale optimization problems this Jacobian is often sparse and the efficient solution of these problems depends on the utilization of this structure [58]. There are two ways to exploit the sparse phenomenon in the Jacobian matrix. One way is to use sparse matrix algorithms, such as sparse LU factorization methods, to speed up the calculation of optimization updates [58]. A second approach to exploit sparsity is to explicitly use the sparse pattern to decouple various optimization functions from various optimization variables, resulting

in optimization decomposition [59]. With decomposition the large scale optimization problem is solved by solving a sequence of small-scale suboptimization problems. The decomposition scheme can be obtained from the sparse pattern of the Jacobian matrix using an automated and adaptive procedure [59], or using a graph theory approach [3]. The resulting suboptimization problems can be formulated as sequential suboptimizations [60], or as a multilevel and parallel suboptimization problem [61].

Chapter 3

A Sparse Optimization Approach to MLP Training

In this chapter, a sparse backpropagation approach based on our work in [22] is proposed for the training of multilayer perceptrons. This approach is formulated by combining the sparse optimization concept from circuit optimization areas, with the backpropagation method which is the most influential training technique for multilayer perceptrons. The sparsity in neural networks is identified from a numerical point of view. We start with a detailed analysis of a 3-layer perceptron, to derive a new set of backpropagation formulae with embedded sparse optimization concept. A generalized sparse training method is then described for multilayer perceptrons, which allows for any number of layers and any number of neurons per layer. A completely new set of parameters that regulate the sparse learning process is introduced. An adaptation procedure for all the new training parameters is proposed. These features are all combined in a systematic and efficient computational algorithm described in this chapter.

3.1 Standard Backpropagation: Dense Training

Consider the 3 layer neural network shown in Figure 3.1.

Figure 3.1: Architecture of the neural network.

The neural network has n_i input neurons, n_h hidden neurons and n_o output neurons. Let w_{ji} and v_{ki} represent the weights in the network. The output of the network, i.e., z_k , $k = 1, 2, \dots, n_o$, can be computed from the input parameters I_i , $i = 1, 2, \dots, n_i$, by a feedforward procedure as

$$z_k = f\left(\sum_{j=0}^{n_h} v_{kj} \cdot y_j\right), \quad k = 1, \dots, n_o, \quad (3.1)$$

$$y_j = f\left(\sum_{i=0}^{n_i} w_{ji} \cdot x_i\right), \quad j = 1, \dots, n_h \quad (3.2)$$

$$x_i = I_i, \quad i = 1, \dots, n_i \quad (3.3)$$

where $y_0 = 1$, $x_0 = 1$ and $f(\cdot)$ is an activation function. For example, the sigmoid activation function is

$$f(\sigma) = \frac{1}{1 + e^{-\sigma}} \quad (3.4)$$

Suppose the training data consists of P pairs of patterns, $\{(\mathbf{I}_p, \mathbf{d}_p), p = 1, \dots, P\}$, where $\mathbf{I}_p$ and $\mathbf{d}_p$ are n_i - and n_o -dimensional vectors representing the inputs and desired

outputs of the MLP, respectively. The purpose of training is to adjust the weights w_{ji} and v_{kj} to minimize the error between the actual and desired outputs:

$$E = \frac{1}{2} \sum_p \sum_{j=1}^{n_o} (z_{pj} - d_{pj})^2 \quad (3.5)$$

where d_{pj} is the j th component of vector $\mathbf{d}_p$ and z_{pj} is the j th output neuron response when the input signal to the MLP is $\mathbf{I}_p$. According to the popular error Backpropagation training algorithm, EBP [19], the basic weight updating during training is along the negative direction of the gradient, i.e.,

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} \quad (3.6)$$

$$\Delta v_{kj} = -\eta \frac{\partial E}{\partial v_{kj}} \quad (3.7)$$

where η is called learning rate. In pattern by pattern training [62], the formulae are

$$\Delta v_{kj} = \eta \delta_{pk}^o y_{pj}, \quad k = 1, \dots, n_o, \quad j = 0, \dots, n_h, \quad (3.8)$$

$$\Delta w_{ji} = \eta \delta_{pj}^h x_{pi}, \quad j = 1, \dots, n_h, \quad i = 0, \dots, n_i \quad (3.9)$$

where

$$\delta_{pk}^o = (d_{pk} - z_{pk}) f'(\sum_{j=0}^{n_h} v_{kj} y_{pj}) = (d_{pk} - z_{pk}) z_{pk} (1 - z_{pk}) \quad (3.10)$$

$$\delta_{pj}^h = (\sum_{k=1}^{n_o} v_{kj} \delta_{pk}^o) f'(\sum_{i=0}^{n_i} w_{ji} x_{pi}) = (\sum_{k=1}^{n_o} v_{kj} \delta_{pk}^o) y_{pj} (1 - y_{pj}) \quad (3.11)$$

The training proceeds epoch by epoch. In each epoch, all training patterns are presented to the MLP for computation of output errors and for backpropagation of the error from the output layer back towards the input layer in order to compute weight updates in Equations (3.8)-(3.11). We refer to this standard training as a dense training technique in order to distinguish it from the proposed sparse training approach described next.

3.2 Sparse Concept in MLP Neural Network Training

From the above formulae, it can be seen that the change of each weight is the product of several factors. If any one of the factors is zero, the resulting change in weights will be zero. For example, Δw_{ji} will be 0 if y_{pj} is 0 or 1, or x_{pi} is 0. This computation is actually carried out in most existing backpropagation training methods, although it produces no change in the corresponding weights. With the sparse matrix concept only nonzero components should be computed. This indicates a potential that much computation in neural network training can be avoided. This phenomenon is clearly present in the case of binary inputs to neural networks, where portions of the input neurons will be zero. In the hidden layers, although many neurons will not be exactly equal to 0 or 1, they will be nevertheless very close to either 0 or 1. Two quantities are close if the difference between them is less than a predefined threshold θ . Therefore Δv_{kj} will be small if z_{pk} is close to d_{pk} , or z_{pk} is close to 0 or 1, or y_{pj} is close to 0. Similarly, Δw_{ji} will be small if y_{pj} is close to 0 or 1, or x_{pi} is close to 0. It is noted that the same weight, e.g., v_{kj} , might need a large change when a different input pattern I_p is presented. In this case insignificant change of v_{kj} as required by one input pattern, is superimposed on significant change of the same v_{kj} required by another input pattern. Therefore, following the sparse optimization concept, small changes in weights can be neglected. This implies that in the backpropagation stage only a subset of neurons need to be processed. Notice that this analysis of sparse concept is based on numerical values of neuron responses. The overall neural network structure, however, is not sparse since all the network weights are retained in the MLP.

For the three layer network, 4 subsets of neurons, S^o , S_1^h , S_0^h and S^i , can be defined for each training pattern:

$$S^o = \{k \mid |d_{pk} - z_{pk}| > \theta^o \text{ and } z_{pk} > \theta^o \text{ and } 1 - z_{pk} > \theta^o\} \quad (3.12)$$

$$S_1^h = \{j \mid 1 - y_{pj} > \theta^h \text{ and } y_{pj} > \theta^h\} \quad (3.13)$$

$$S_0^h = \{j \mid y_{pj} > \theta^h\} \quad (3.14)$$

$$S^i = \{i \mid x_{pi} > \theta^i\} \quad (3.15)$$

where θ^o , θ^h and θ^i are thresholds for output layer, hidden layer and input layer, respectively. These subsets provide the sparse patterns that can be used in sparse optimization [59].

The updates of weights will be necessary only for these subsets of neurons:

$$\Delta v_{kj} = \eta \delta_{pk}^o y_{pj}, \quad k \in S^o \text{ and } j \in S_0^h \quad (3.16)$$

$$\Delta w_{ji} = \eta \delta_{pj}^h x_{pi}, \quad j \in S_1^h \text{ and } i \in S^i \quad (3.17)$$

The concept of sparse training is further illustrated through a fictitious example in Figures 3.2 and 3.3. In Figure 3.2, the feedforward path in the dense technique is through all the neurons and all the network links (weights). For the sparse technique, the feedforward path is through only the neurons whose outputs are non-zero. In Figure 3.3, the backpropagation path in the dense technique is through all the neurons and all the network links. For the sparse technique, the backpropagation path is through only the neurons whose outputs are non-zero and non-unity. Notice that for different training patterns, the sparsity in the neural network can be different, as represented by training patterns 1 and 2 in the figures. Clearly, the computation in the sparse technique is much less than that required in the dense technique.

(a)

$$S_0^h = \{ 2, 3 \}$$

$$S_0^i = \{ 2, 3 \}$$

(b)

$$S_0^h = \{ 1, 3 \}$$

$$S_0^i = \{ 3, 4 \}$$

(c)

Figure 3.2: Feedforward path in (a) dense training approach, and (b)- (c) sparse training approach. The feedforward path in the sparse training approach is different depending on the training data as shown in (b) for training pattern 1, and (c) for training pattern 2.

(a)

error: 0.08 0.4

$$S_1^o = \{2\}$$

$$S_0^h = \{2, 3\}$$

$$S_1^h = \{3\}$$

$$S_0^i = \{2, 3\}$$

(b)

error: 0.07 0.2

$$S_1^o = \{2\}$$

$$S_0^h = \{1, 3\}$$

$$S_1^h = \{1\}$$

$$S_0^i = \{3, 4\}$$

(c)

Figure 3.3: Backpropagation path in (a) dense training approach, and (b)-(c) sparse training approach. The backpropagation path in the sparse training approach is different depending on the training data as shown in (b) for training pattern 1, and (c) for training pattern 2.

Let a set of sparse factors be defined as

$$\rho^o = 1 - \frac{|S^o|}{n_o} \quad (3.18)$$

$$\rho_1^h = 1 - \frac{|S_1^h|}{n_h} \quad (3.19)$$

$$\rho_0^h = 1 - \frac{|S_0^h|}{n_h} \quad (3.20)$$

$$\rho^i = 1 - \frac{|S^i|}{n_i} \quad (3.21)$$

where $|S|$ represents the number of the elements in the set S . The ideal speed up ratio β , i.e., the number of updates of weights in dense technique vs. that in sparse technique, is

$$\beta = \frac{1}{P} \sum_p \frac{n_i \times n_h + n_h \times n_o}{|S^i| |S_1^h| + |S_0^h| |S^o|} = \frac{1}{P} \sum_p \frac{n_i + n_o}{(1 - \rho^i)(1 - \rho_1^h)n_i + (1 - \rho_0^h)(1 - \rho^o)n_o} \quad (3.22)$$

A simplified estimation of the speedup factor counting other computations common to both techniques (represented by γ) could be $\frac{1}{(1-\rho)^2(1-\gamma)+\gamma}$, where ρ represents the average sparsity.

Now consider training with momentum term. Since the weights in neural networks are not updated consecutively, i.e., subsets S^o, S_1^h, S_0^h and S^i change with patterns, the updating formula with momentum term present requires special considerations. Let α be the momentum factor. Let q_{kj}^o and q_{ji}^h be the number of training patterns presented between two consecutive updates of weights v_{kj} and w_{ji} , respectively. The effect of momentum due to sparse backpropagation can be accumulated through

$$v_{kj} = v_{kj} + (\alpha + \alpha^2 + \dots + \alpha^{q_{kj}^o-1})(\Delta v_{kj})_{old} = v_{kj} + \frac{\alpha(1 - \alpha^{q_{kj}^o-1})}{1 - \alpha}(\Delta v_{kj})_{old} \quad (3.23)$$

$$(\Delta v_{kj})_{new} = \alpha^{q_{kj}^o}(\Delta v_{kj})_{old} + (1 - \alpha)\eta\delta_{pk}^o y_{pj} \quad (3.24)$$

$$v_{kj} = v_{kj} + (\Delta v_{kj})_{new} \quad (3.25)$$

$$(\Delta v_{kj})_{old} = (\Delta v_{kj})_{new} \quad (3.26)$$

Similarly,

$$w_{ji} = w_{ji} + (\alpha + \alpha^2 + \dots + \alpha^{q_{ji}^h - 1})(\Delta w_{ji})_{old} = w_{ji} + \frac{\alpha(1 - \alpha^{q_{ji}^h - 1})}{1 - \alpha}(\Delta w_{ji})_{old} \quad (3.27)$$

$$(\Delta w_{ji})_{new} = \alpha^{q_{ji}^h}(\Delta w_{ji})_{old} + (1 - \alpha)\eta\delta_{pj}^h x_{pi} \quad (3.28)$$

$$w_{ji} = w_{ji} + (\Delta w_{ji})_{new} \quad (3.29)$$

$$(\Delta w_{ji})_{old} = (\Delta w_{ji})_{new} \quad (3.30)$$

3.3 Sparse Backpropagation in General Multilayer Perceptrons

In the previous section, a sparse training approach is derived through a detailed analysis of the sparse phenomenon in a 3-layer perceptron environment. In this section, we will describe the sparse training technique for the general multilayer perceptron case, i.e., MLP with any number of layers and any number of neurons per layer.

Figure 3.4 shows a general MLP structure. Suppose the total number of layers is L . The input layer is layer 1, and the output layer is layer L . Let the number of neurons in layer l be n_l . Let w_{ij}^l represent the weight for the link between neuron j at layer $l - 1$ and neuron i at layer l . Let I_i represent the i th input parameters to the MLP. Let y_i^l be the output of neuron i at layer l , which can be computed according to the standard dense MLP approach as

$$y_i^l = f\left(\sum_{j=0}^{n_{l-1}} w_{ij}^l y_j^{l-1}\right) \quad i = 1, \dots, n_l, \quad l = 2, \dots, L \quad (3.31)$$

Figure 3.4: General Multilayer Perceptrons.

where $f(\cdot)$ is a sigmoid function, and

$$y_i^1 = I_i, \quad i = 1, 2, \dots, n_1.$$

$$y_0^l = 1, \quad l = 1, 2, \dots, L-1$$

To use the sparse training concept for the L layer MLP neural network, we define $2L$ subsets of neurons, i.e., $S_1^l, S_0^l, l = 1, \dots, L$, for each training pattern. For each of the hidden layers, $l = 2, \dots, L-1$, the 2 subsets of neurons are defined as

$$S_1^l = \{j | 1 - y_{pj}^l > \theta^l \text{ and } y_{pj}^l > \theta^l\} \quad (3.32)$$

$$S_0^l = \{j | y_{pj}^l > \theta^l\} \quad (3.33)$$

where θ^l is the sparse threshold for layer $l, l = 2, \dots, L-1$.

Since there is no need to feedforward the signal after the output layer and to backpropagate error beyond the input layer, the subsets for these two layers can be defined as

- input layer ($l = 1$):

$$S_1^1 = \emptyset \quad (3.34)$$

$$S_0^1 = \{j | y_{pj}^1 > \theta^1\} \quad (3.35)$$

where $y_{pj}^1 = I_{pj}$.

- output layer ($l = L$):

$$S_1^L = \{j | |y_{pj}^L - d_{pj}| > \theta^L \text{ and } 1 - y_{pj}^L > \theta^L \text{ and } y_{pj}^L > \theta^L\} \quad (3.36)$$

$$S_0^L = \emptyset \quad (3.37)$$

The sparse concept is applied to both feedforward and backpropagation computations. The feedforward computation exploiting sparsity is

$$y_i^l = f\left(\sum_{j \in S_0^{l-1}} w_{ij}^l y_j^{l-1}\right) \quad i = 1, \dots, n_l, \quad l = 2, \dots, L \quad (3.38)$$

During backpropagation, the sparse optimization concept allows the updates of weights to be performed only for these subsets of neurons:

$$\Delta w_{ij}^l = \eta \delta_{pi}^l y_{pj}^{l-1}, \quad i \in S_1^l, \quad j \in S_0^{l-1}, \quad l = 2, \dots, L \quad (3.39)$$

where δ_{pi}^l is a local gradient defined also using the sparse concept as

$$\delta_{pi}^L = (d_{pi} - y_{pi}^L) y_{pi}^L (1 - y_{pi}^L) \quad (3.40)$$

$$\delta_{pi}^l = \left(\sum_{k \in S_1^{l+1}} \delta_{pk}^{l+1} w_{ki}^{l+1} \right) y_{pi}^l (1 - y_{pi}^l) \quad \text{for } i \in S_1^l \text{ and } l = 2, \dots, L-1 \quad (3.41)$$

Let sparse factors be defined as

$$\rho_1^l = 1 - \frac{|S_1^l|}{n_l} \quad (3.42)$$

$$\rho_0^l = 1 - \frac{|S_0^l|}{n_l} \quad (3.43)$$

where $l = 1, \dots, L$, and $|S|$ represents the number of elements in set S .

The ideal speed up ratio β of the sparse training over dense training is estimated as following,

$$\begin{aligned} \beta &= \frac{1}{P} \sum_p \frac{\sum_{l=1}^{L-1} n_l n_{l+1}}{\sum_{l=1}^{L-1} |S_0^l| |S_1^{l+1}|} \\ &= \frac{1}{P} \sum_p \frac{\sum_{l=1}^{L-1} n_l n_{l+1}}{\sum_{l=1}^{L-1} (1 - \rho_0^l)(1 - \rho_1^{l+1}) n_l n_{l+1}} \end{aligned} \quad (3.44)$$

A simplified estimation of the speed up factor counting other computations common to both techniques (represented by γ) could be $\frac{1}{(1-\rho)^2(1-\gamma)+\gamma}$, where ρ represents the average sparse factor.

The momentum effect on the weights can be derived similarly to that in the three layer MLP neural network case. Let q_{ij}^l be the number of training patterns presented between two consecutive updates of weights w_{ij}^l , $q_{ij}^l \geq 1$. The effect of momentum term due to sparse backpropagation for the general multilayer case can be accumulated through

$$w_{ij}^l = w_{ij}^l + (\alpha + \alpha^2 + \dots + \alpha^{q_{ij}^l - 1})(\Delta w_{ij}^l)_{old} = w_{ij}^l + \frac{\alpha(1 - \alpha^{q_{ij}^l - 1})}{1 - \alpha}(\Delta w_{ij}^l)_{old} \quad (3.45)$$

$$(\Delta w_{ij}^l)_{new} = \alpha^{q_{ij}^l}(\Delta w_{ij}^l)_{old} + (1 - \alpha)\eta\delta_{pi}^l y_{pj}^{l-1} \quad (3.46)$$

$$w_{ij}^l = w_{ij}^l + (\Delta w_{ij}^l)_{new} \quad (3.47)$$

$$(\Delta w_{ij}^l)_{old} = (\Delta w_{ij}^l)_{new} \quad (3.48)$$

3.4 Algorithms for Sparse Backpropagation Training of General MLP

In this section, we propose algorithms that incorporate the sparse training approach described in previous sections through systematic and efficient computational steps. Two algorithms that have been developed, implemented and tested are described in this section. Algorithm 1 is based on the sparse feedforward and sparse backpropagation methods described in the previous section. Algorithm 2 introduces an advanced concept into sparse training, that of adaptively controlled refreshment of sparse patterns. Algorithm 1 will be referred to as SEBP1 (Sparse Error Backpropagation Algorithm 1) and Algorithm 2 will be referred to as SEBP2 (Sparse Error Backpropagation Algorithm 2).

3.4.1 Algorithm 1 (SEBP1)

Step #1: Set up the MLP neural network, i.e. number of layers, L , and number of neurons per layer, $n_l, l = 1, \dots, L$. Initialize the weights to be random values.

Step #2: Initialize all the training parameters, e.g., learning rate η , momentum factor α , sparse threshold $\theta^l, l = 1, \dots, L$, threshold for training error ϵ, ϵ' , and the maximum number of epochs MAX_N_Epochs. Set $n_{epochs} = 1$.

Step #3: Create subset S_0^1 of input layer for each training pattern according to Equation (3.35). Take the first pattern.

Step #4:

Step #4a: Perform sparse MLP feedforward for the input layer, utilizing S_0^1 in feedforward computation from the input layer to the next layer. Perform sparse feedforward for the subsequent layers following Equation (3.38). Create subsets S_0^l and S_1^l for $l = 2, \dots, L$ according to Equations (3.32), (3.33) and (3.36).

Step #4b: calculate the sum of squared error between desired outputs and the actual outputs of the MLP and then accumulate the error over that from previous training patterns for the current epoch.

Step #4c: Perform the sparse error backpropagation and simultaneously perform the sparse weight updating utilizing the subsets S_0^l and S_1^l for $l = 2, \dots, L$ according to equation (3.39).

Step #5: If all the training patterns have been used in the current epoch, then go to step #6.

Otherwise, take the next training pattern, go to step #4.

Step #6: Check stop criteria: If the error is below ϵ or slope of the error has been consecutively below ϵ' or $n_{epochs} = MAX_N_Epochs$, then save the weights and stop.

Step #7: Adapt learning rate η . (Detailed description of the adaptation procedure is given in Section 3.7.1).

Step #8: Adapt sparse thresholds, $\theta^1, \theta^2, \theta^3, \dots, \theta^L$. (Detailed description of the adaptation procedure is given in Section 3.7.2).

Step #9: If cross validation is not needed, go to Step #10.

otherwise, perform cross validation.

if the current cross validation performance is better than previously the best one, then

save the weights,

set current performance as the best one,

go to step #10.

else,

if performance goes down consecutively, stop.

Step #10: If after several epochs training error doesn't go down significantly and the error is still above a certain level (which indicates training is still in the premature level), then perform random perturbation.

Step #11: Let $n_{epochs} = n_{epochs} + 1$, take the first pattern, and go to Step #4.

In the above algorithm, the outputs of all neurons at all layers except the input layer are computed during every epoch. In this case, the sparse feedforward formula

is

$$y_i^l = f\left(\sum_{j \in S_0^{l-1}} w_{ij}^l y_j^{l-1}\right) \quad i = 1, \dots, n_l, \quad \text{for } l = 2, \dots, L \quad (3.49)$$

i.e., the formula is computed for all neurons, $i = 1, \dots, n_l$, for $l = 2, \dots, L$. In this way the subsets S_1^l and S_0^l can be updated at every epoch, i.e., the sparse patterns are refreshed every epoch. Notice that the sparse concept is based on neuron output values instead of neural network topology. Updating the subsets of S_1^l and S_0^l is to ensure that the sparse pattern used for both feedforward and backpropagation is always accurate for the given sparse thresholds.

3.4.2 Algorithm 2 (SEBP2)

In Algorithm 2 to be proposed below, we will introduce a new training concept, where the sparse patterns are not updated in consecutive epochs. In this algorithm, the feedforward computation is not performed for all neurons as in Equation (3.49). The new feedforward scheme is

$$y_i^l = f\left(\sum_{j \in S_0^{l-1}} w_{ij}^l y_j^{l-1}\right) \quad i \in S_0^l, \quad \text{for } l = 2, \dots, L \quad (3.50)$$

The computation only for neurons in the subset S_0^l instead of for all neurons is adequate, since in sparse feedforwarding subsequent layers and in future sparse backpropagation, it is only these subsets of neurons that are needed anyway. In this way, further exploitation of sparsity is achieved and the feedforward computation becomes completely a sparse procedure. Therefore we refer to Equation (3.50) as fully sparse feedforward, and Equation (3.49) as partially sparse feedforward. The ultimate benefit of the fully sparse approach is to achieve further speedup in computation.

However, the fully sparse approach requires the knowledge of the sparse pattern, i.e., subsets S_0^l , which can not be computed until after feedforwarding beyond the

present layer. In order to solve this dilemma, we will use the sparse pattern obtained from the previous epoch. Therefore, in Algorithm 2, we will keep the sparse pattern, i.e., subsets S_0^l and S_1^l , unchanged for several consecutive epochs. To avoid the risk of using totally inaccurate sparse patterns, we implement a mechanism where the sparse patterns are refreshed after several epochs. This however, will have implications on learning rate adaptation. For example, the normal learning rate adaptation procedure will not be reliable when sparse pattern refreshment is carried out and a sudden change in training curve occurs. How to control the sparse pattern refreshment intervals will also affect the training.

To clearly present this new algorithm, we first show the overall flow diagram and describe the overall computation steps, followed by detailed descriptions of the various training parameters in subsequent sections.

This new algorithm is called Algorithm 2 or SEBP2 (Sparse Error Backpropagation Algorithm 2), whose flow diagram is shown in Figure 3.5.

Algorithm 2:

Step #1: Same as in Algorithm 1.

Step #2: Initialize all the training parameters, e.g., learning rate η , momentum factor α , sparse threshold θ^l , $l = 1, \dots, L$, threshold for training error ϵ , ϵ' , maximum number of epochs MAX_N_Epochs, and sparse pattern refreshment interval. Set $n_{epochs} = 1$.

Step #3: Same as in Algorithm 1.

Step #4a: If refreshment of sparse patterns is needed,

perform sparse MLP feedforward for the input layer, utilizing S_0^1 in

feedforward computing from the input layer to the next layer,
 perform partially sparse feedforward for the subsequent layers
 following Equation (3.38),
 create subsets S_0^l and S_1^l for $l = 2, \dots, L$ according to Equations
 (3.32), (3.33) and (3.36).

else

use the existing sparse patterns, i.e. S_0^l and S_1^l , $l = 2, \dots, L$, to
 perform fully sparse feedforward following Equation (3.50).

Step #4b to 4c: Same as in Algorithm 1.

Step #5 to 6: Same as in Algorithm 1.

Step #7: Learning rate adaptation: Two different adaptation criteria are used depending upon whether there is a refreshment of sparse pattern or not. Detailed description is provided in Section 3.7.1.

Step #8-9: Same as in Algorithm 1.

Step #9b: Sparse refreshment interval adaptation. Detailed description is given in Section 3.7.3.

Step #10-11: Same as in Algorithm 1.

In this second algorithm, a further speed-up can be achieved by reusing the subsets S_0^l and S_1^l of each training pattern for several consecutive epochs. The computational effort of deriving the sparse patterns and deleting the previous sparse patterns is saved, in addition to the computational savings from a fully sparse feedforward.

However the sparse patterns from previous epochs may not be always correct for the present epoch. The use of previous sparse patterns therefore may cause the learning curve not going down smoothly. The corresponding issues will be discussed in the following sections.

3.5 Stop Criteria

The stop condition used in our algorithms is a combination of 4 criteria. The first criterion used is to check if the training error is already below a given threshold ϵ .

The second criterion is from the cross validation test. In this test, the performance of the neural network on a validation data set, which is separate from training data, is checked. If the performance is getting poorer, it is time to stop training to avoid overlearning. In the actual implementation, it is unnecessary to check the validation performance at every epoch. In our algorithm, the interval of cross validation tests is specified by the user. The decision to stop training is obtained from looking at several (e.g. 3) consecutive validation tests.

To avoid the training program to run infinitely long time, and to allow users to save and assess training results, two more stop criteria are also included. The third criterion used is to check if the derivative of error curve is already nearly zero. The fourth criterion is to check if a preset limit on the maximum number of epochs is reached.

3.6 Random Perturbation

In reality, it is easy for the neural network training to be trapped at a premature saturation point. In this case the instantaneous sum of squared error E in the training process remains almost constant for some period of time during the learning process. This is due to the fact that some of the neurons are in the wrong saturation region of the neuron activation function. In the sparse-concept based training algorithms, since we emphasize the strong activation of some neurons by ignoring the weak activation of other neurons, this phenomenon can be observed also from the selection of sparse patterns, i.e., subsets S'_0 and S'_1 . When such phenomenon occurs, it is a good indication that the current internal representation of the MLP is not good for the learning. Instead of letting the neural network itself to find the correct representation, an external force is introduced, i.e., random perturbation [31] [32]. This perturbation in reality yields an opportunity to draw the neurons out of the wrong saturation region so that they can find their new and more correct ways. This helps optimization to jump out of local minimum. Experimental results show that after perturbation the error can go down very fast and this perturbation usually leads to a better performance.

3.7 Sparse Training Parameters and Their Adaptation

The use of sparse training technique necessitates the introduction of several completely new parameters that regulate the sparse learning process. These new parameters can become adaptive in an advanced sparse training algorithm. Also in advanced sparse training, adaptation of some conventional training parameters needs to be modified. This section describes these training parameters and their adaptation

schemes which are needed as a result of using the sparse training technique.

3.7.1 Learning Rate Adaptation

The adaptation for this parameter has been an important way to speedup MLP training [26] [9] [29]. The proposed sparse training algorithms can be applied on top of the existing adaptation schemes used in conventional dense training.

Suppose E represents the total error of training at a particular epoch. A basic adaptation scheme, such as the one used in Matlab [25], is to increase or decrease the learning rate by a certain percentage with respect to its previous value according to if E goes down or goes up from epoch to epoch. Based on this scheme, a learning rate adaptation procedure for the sparse training algorithm can be developed as following.

Let E_{new} be the error E at the present epoch.

If sparse pattern refreshment was done in the current epoch, then

$E_{old} = E$ at the epoch of the previous refreshment.

else

$E_{old} = E$ at the previous epoch.

if $E_{new} < E_{old}$, then

$\eta = \eta * R_{inc}$

else

if $E_{new} > E_{old}(1 + \tau)$, then $\eta = \eta * R_{dec}$, and discard the weights.

where R_{dec} and R_{inc} are the ratios to decrease or increase learning rate η . Both values can be set by the user. For example, $R_{dec} = 0.7$, $R_{inc} = 1.05$. τ is a small value, e.g., $\tau = 0.01$.

This adaptation procedure was applied to both Sparse Algorithms 1 and 2. In the case of Algorithm 1, the sparse pattern is refreshed in every epoch and the adaptation procedure becomes identical to that in the dense training case.

3.7.2 Sparse Threshold

This parameter is created for the sparse-concept based training algorithms. It is used to control the formulation of subsets S_0^l and S_1^l for $l = 1, \dots, L$, i.e., sparse patterns of the MLP neural network. Each layer in the MLP has its own threshold, resulting in a set of parameters $\theta^1, \theta^2, \dots, \theta^L$. If the input to the MLP is in the binary format, θ^1 can be set to any number between (0,1) because there is a very clear difference between input "0" and input "1". If the input is not in the binary format, this θ^1 is set by the user according to the application problem. For the other parameters $\theta^2, \dots, \theta^L$, there is no distinction between binary and continuous cases. In our algorithm, all the sparse threshold parameters $\theta^1, \theta^2, \dots, \theta^L$ are allowed to be adaptive. According to our experience in implementing this algorithm, the best result is usually obtained when the thresholds for hidden layers are fixed at a slightly large value, e.g. 0.1, while the threshold for the output layer is adaptively reduced from the initial value, e.g. 0.1. This is because that when the training procedure continues, the final outputs of the MLP approach the desired outputs more and more closely and therefore with larger threshold the subset S_1^L would be empty and no error backpropagation would occur. A larger value of θ^L at the beginning can be used to speed up the training. For hidden layers, larger value of θ^l can be used to emphasize the strong neuron activations, and as long as the whole network can reach an acceptable quality, larger value can be used to speed up training. This effect is similar to sparse circuit optimization where weak couplings between design

variables and design specifications are ignored. The experiential results show that the sparse-concept based algorithms can obtain the same and sometimes even better quality of the trained network. The following actual implementation of the adaptation procedure reflects the above description.

if (error E doesn't go down significantly for several epochs and there is no recent random perturbation and threshold reduction)

if ($\theta^L > level_o$)

$$\theta^L = \theta^L * T_{dec_ratio}$$

if (error $E < level_e$)

for($l = 1, \dots, L - 1$)

if ($\theta^l > level_h$)

$$\theta^l = \theta^l * T_{dec_ratio}$$

where T_{dec_ratio} is the decreasing ratio, $level_o$ and $level_h$ are the threshold levels to control if it is necessary to further reduce output layer and non-output layer sparse thresholds, respectively. $level_e$ is the error level to indicate if it is necessary to reduce non-output layer sparse thresholds. So before the error reaches $level_e$, the non-output layer sparse thresholds are kept at their larger initial values and thus output layer sparse threshold is given the preference to reduce. All these parameters can be set by the user and the suggested value for T_{dec_ratio} is 0.1. As a special case, if all the sparse thresholds are set to zero, the proposed sparse training technique becomes identical to conventional dense training.

3.7.3 Sparse Pattern Refreshment Interval

This is a new parameter created for the sparse training Algorithm 2. This algorithm is aimed at speeding up the training further beyond that in sparse training Algorithm 1. It is based on the idea that if the sparse pattern is stable across several epochs, more computational efforts can be saved by reusing sparse patterns for several epochs. According to our observation, the sparse patterns at the beginning of training change significantly between epochs while at the end of training they are more stable. To make the idea of reusing sparse patterns more suitable for the MLP training, it is needed to make the interval for sparse pattern refreshment adaptatively changed. At the beginning of the training, set the refreshment interval to 1. In this case Algorithm 2 performs in the same way as Algorithm 1, i.e. recomputing subsets S_0^l and S_1^l at each epoch. Later in the training, the refreshment interval is gradually increased to speed up the training. The actual implementation is as following:

if (current validation test result is better than the previous one)

$$interval = interval + 1$$

In this way, a trade off is obtained between maintaining reliable sparse patterns using refreshment and training speedup by fully sparse feedforward and backpropagation.

3.8 Conclusion

A sparse backpropagation training approach has been proposed for the training of multilayer perceptrons. The sparse concept is applied to both feedforward and backpropagation phases, yielding significant computational savings over the conventional dense training approach. A new set of parameters that regulate the sparse learning

process is introduced. An adaptation procedure for the new training parameters is proposed. Two algorithms are proposed incorporating the sparse training approach into systematic and efficient computational steps.

The advantage of Algorithm 1 is its maintenance of sparse patterns which are always valid for the given sparse threshold. The learning procedure is expected to be more smooth than that of Algorithm 2 where sparse patterns are not always up to date. In other words, the total number of epochs required by Algorithm 1 to train the network adequately may be less than that in Algorithm 2.

The advantage of Algorithm 2 is its speed per epoch. Since Algorithm 2 uses fully sparse feedforward, the computation for each epoch in Algorithm 2 should be less than that in Algorithm 1.

Both algorithms have been implemented and tested by application examples. Both algorithms provide significant speedup in training over the dense training approach. The results from the application examples are presented in Chapters 4 and 5. With many training parameters, newly introduced plus conventional, the coordination among them becomes very important, such as which parameter should be given the preference to change at certain point of the training process. Actual values for training parameters and their adaptation are further presented in the application examples. The new sparse training algorithms are implemented in a computer program written in C++ under UNIX. The instruction for using the sparse MLP neural network program is included in the thesis under Appendix A.

Figure 3.5: Flow diagram of sparse-concept based training Algorithm 2.

Chapter 4

Application to Speech Recognition

In this chapter the proposed sparse MLP training technique is applied to speech recognition, which is one of the existing areas for neural networks already under intensive investigation by many researchers. It is also a challenging application for neural networks. In the previous chapter, the sparse concept and the derivation of sparse training methods and algorithms were mainly based on conceptual analysis and illustrations of fictitious examples. In this chapter, we will examine several critical issues, such as whether the envisaged sparsity exists or not in a practical application; whether the proposed training algorithm will train a realistic neural network to a satisfactory solution; how sparse a realistic neural network may be; how much computational speedup can the proposed algorithm achieve. The success of the new training algorithm in this existing neural network area will be useful to confirm the validity of the proposed approach as a viable alternative to conventional neural network training.

The chapter starts with the introduction to the speech recognition area. A speech recognition software system was developed [63] for the application of sparse MLP training. The study of sparse training for isolated word recognition is provided.

Experimental results with Texas Instruments speech data base TI20 and TI46 are described.

4.1 Introduction

The use of neural networks for speech recognition has been an important topic for many years [64], [65]. Neural networks with their ability to learn and to generalize, is one of the attractive vehicles for pattern recognition where precise relationships in the patterns do not exist. However despite the serious effort of research over many years in the past, speech recognition remains one of the most difficult tasks of temporal pattern recognition today. Many factors contributed to the problem, among them are, e.g., spectral variations due to inter-speaker and intra-speaker variations, temporal variations in speech signals, noise, confusability and size of vocabulary, coarticulation, etc. [66].

Neural network is an attractive vehicle for speech recognition and is currently one of the active areas of research in the speech recognition area [67], in addition to other techniques such as those based on Hidden Markov Models (HMM) [68]. Neural networks are particularly appealing with their ability of discriminant-based learning, i.e., models are trained to minimize the error rate while maximizing the distance between the correct pattern and its rivals. Another attractive feature of neural networks is that they can be applied with no statistical assumptions on the input feature parameters. In addition neural networks have a very flexible architecture which can easily accommodate contextual inputs and feedback [69].

One of the challenges for neural network speech recognizers is due to their inadequate capability to handle temporal variation and sequential information which are

inherent in speech signals. This problem has received attention from many researchers and a number of techniques have been developed. The Time-Delay Neural Network (TDNN) [70], for example, is a very attractive architecture for learning the temporal relationships in acoustic events with reduced sensitivity to shifts in time. Recent advance in this approach is the Multi-State TDNN and the addition of a nonlinear time alignment procedure (DTW) to further improve the time-alignment capability of the model [71] [72]. Another popular approach is the hybrid system of neural networks and Hidden Markov Models (HMM), e.g., [69] [73]. In this case the neural network can be used, e.g., to approximate functions for computing acoustic parameters to be used as observations by HMM [73]. In these approaches, processing temporal information is partly or mainly done using mechanisms other than the neural network, i.e., using DTW in [71] and HMM in [69] [73]. A drastic and interesting solution is to specifically construct a neural network to virtually emulate the mechanism of Hidden Markov Models e.g., [64] [74], fully taking advantage of the powers of HMM in speech recognition.

Neural network techniques for speech recognition with improved temporal processing capability without the explicit use of DTW or HMM concepts have also been studied. Among these approaches are, for example, the combined multilayer perceptron/self-organizing maps (SOM) [75] where two separate SOM are used to split the overall signal into first and second halves, the sequential competitive avalanche field where a constellation of activity develops over selected nodes by letting previously fired neurons decay slowly [62], the use of recurrent neural networks which processes signals in both space and time domains [76] [77], and the use of high-order temporal neural networks [78].

4.2 Speech Recognition System

The speech recognition system we developed follows the neural network approach. Figure 4.1 shows the structure of a combined multi-codebook/MLP (Multilayer Perceptron) neural network speech recognition system that is used in our work and is targeted for isolated word recognition. The basic structure of the system is based on the multi-stage Neural Network approach of [75] [78]. The signal preprocessing is based on the cepstral multi-codebook approach of [79]. The overall system is divided into four levels. In level 1, the basic preprocessing of the speech signals were performed. The speech signal waveforms were sampled, and pre-emphasized. The sampled waveforms were blocked into frames from which LPC (Linear Predictive Coding) parameters were computed. In level 2, the basic LPC parameters were used to derive a set of feature parameters. The resulting feature space defined by these parameters will provide the basic measure of the similarity or distance between various speech signals. In this work, we follow the multi-codebook approach of [79], with the addition of higher order cepstral parameters as used in [80]. The resulting feature parameters consist of four sets of parameters, i.e., cepstral parameters, cepstral derivative parameters, second order cepstral derivative parameters, and energy parameters (energy and energy derivative). In level 3, the feature space is quantized into codebooks. A total of four codebooks are used corresponding to the four sets of feature parameters, i.e., cepstral codebook, delta-cepstral codebook, delta-delta cepstral codebook, and energy and delta-energy codebook. In level 4 is a multilayer perceptron, where the pattern recognition task is performed and the result of speech recognition is provided. The inputs of the MLP are the codes from the codebooks, and the outputs of the MLP are the selection of the words recognized.

In the next few sections, a more specific and detailed description of the various parts of the system and their training is provided.

4.3 Speech Signal Preprocessing

The proposed sparse training technique is applied to isolated word recognition. We used the Texas Instrument TI20 real speech data [75] which contains 10 digits and 10 command words and TI46 data base which includes TI20 and 26 alphabets. Speech signals of the data have been low-pass filtered at $6kHz$ and sampled at $12.5kHz$ with 12 bit resolution. In our preprocessing stage, they were pre-emphasized with a filter whose transfer function is $1 - 0.95z^{-1}$. The waveform was then blocked into frames and multiplied by a Hamming window with length and shifts being $16ms$ and $10ms$ respectively. From these smoothed speech samples 12 LPC-derived cepstral parameters per frame were computed together with 12 cepstral derivative parameters, 12 second order cepstral derivative parameters, energy and energy derivative per frame. From these parameters four codebooks were generated, i.e., cepstral codebook (size 144), delta-cepstral codebook (size 144), delta-delta-cepstral codebook (size 144) and energy codebook (size 36). The total number of codes from the four codebooks combined is 468.

4.4 Clustering Algorithm for Codebook Development

Unsupervised learning algorithm plays a central part in models of neural computation. K-means Clustering algorithms, a type of unsupervised learning algorithms, have been used in many application areas. We develop an improved k-means algo-

rithm [63] in this section for optimal partition which can achieve better variation equalization than standard binary splitting algorithms. The proposed clustering algorithm was incorporated in level 3 of the combined multicodebook/MLP neural network speech recognition system to train the LPC based codebooks. It achieved smaller variation of the variances of clusters than that from the standard binary splitting algorithm.

4.4.1 Introduction

K-means Clustering algorithms, the most commonly used competitive learning algorithms, have been applied to many applications such as speech recognition as the front end feature extractor [68] [79], data compression as vector quantizer [81] [82], and so on. In neural computation, this kind of unsupervised learning algorithm is also an important part. However, the partition results of traditional k-means algorithms such as generalized Lloyd clustering algorithm [68] rely on the initial positions of reference vectors heavily. Research in this area have resulted in various advanced algorithms, e.g. starting from the region of all the input vectors and performing binary splitting [68] [83], biasing the distance measure towards the regions with smaller variations to achieve variation-equalization [84]. In typical binary splitting type algorithms, however, the partition result can still be easily trapped in a solution which is only local minimum according to the optimal k-means partition criterion which states that all the regions should have the same variations. In this section, we develop an improved k-means algorithm for optimal partition which can achieve better variation-equalization than that from standard binary splitting algorithms.

4.4.2 Algorithm

The algorithm we developed is a nonbinary splitting algorithm [63] which starts from the region of all the input vectors $\{X_p | p = 1, \dots, P\}$. The partition regions are then split stage by stage into K regions, where K is an application dependent parameter. Instead of binary splitting at each stage, only the regions with relatively larger variance are to be split. In such a way, the algorithm achieves final solution with a partition in which all the regions have similar variance.

Let set $R_i = \{p | X_p \in \text{the } i\text{th region}\}$ represent the $i\text{th}$ region and C_i be the centroid of the $i\text{th}$ region. Let set $S = \{i | R_i \text{ to be split}\}$. Let $n_regions$ be the current number of regions, n_sweeps be the number of sweeps passed at the refinement stage and θ be the threshold to select the regions to be split. There are also some predefined constants, i.e. γ as the splitting rate, ϵ as the threshold criterion to exit refinement stage and MAX_N_SWEEPS as the largest number of sweeps allowed at the refinement stage. The proposed nonbinary splitting algorithm is given below.

initialization stage

step#1: set $n_regions = 1, R_1 = \{p | p = 1, \dots, P\}$,

$S = \{1\}$,

$C_1 = (\sum_{p=1}^P X_p) / P$;

split stage

step#2: for(all the $i, i \in S$)

split the region R_i into two regions:

$n_regions = n_regions + 1$;

compute centroid of region $R_{n_regions}$

$$C_{n_regions} = C_i \times (1 - \gamma);$$

compute centroid of region R_i

$$C_i = C_i \times (1 + \gamma);$$

$n_sweeps = 0;$

for($i = 1$ to $n_regions$)

$$R_i = \emptyset;$$

refinement stage

step#3: $n_sweeps = n_sweeps + 1;$

for($p = 1, \dots, P$)

find the nearest centroid C_j to X_p

among $\{C_i | i = 1, \dots, n_regions\};$

assign X_p to region $R_j;$

step#4: for($i = 1$ to $n_regions$)

update the centroid of region R_i

$$(C_i)_{new} = (\sum_{p \in R_i} X_p) / |R_i|;$$

where $|R_i|$ represents the number of input vectors in set R_i .

step#5: find the largest movement of centroid

$$max_move = \max_{i=1}^{n_regions} \|(C_i)_{new} - C_i\|$$

step#6: if ($max_move > \epsilon$ and $n_sweeps < MAX_N_SWEEPS$)

for($i = 1$ to $n_regions$)

$$C_i = (C_i)_{new};$$

$$R_i = \emptyset;$$

go to step#3.

decision stage

step#7: if ($n_regions = K$) then stop;

step#8: for($i = 1$ to $n_regions$)

 calculate the variance ν_i of region R_i ;

step#9: calculate the average value and maximum value of the variances:

$$\bar{\nu} = \left(\sum_{i=1}^{n_regions} \nu_i \right) / n_regions$$

$$\nu_{max} = \max_{i=1}^{n_regions} \nu_i$$

step#10: set threshold $\theta = \frac{1}{2}(\bar{\nu} + \nu_{max})$

$S = \emptyset$;

 for($i = 1$ to $n_regions$)

 if ($\nu_i > \theta$)

 put region R_i into set S ;

 if ($n_regions + |S| > K$)

$S = \{R_i | \nu_i = \nu_{max}\}$

 go to step#2;

4.4.3 Examples

The algorithm was used to train four codebooks of the Texas Instrument speech data TI20, namely, cepstral codebook, delta-cepstral codebook, delta-delta-cepstral codebook and energy and energy derivative codebook. Tables 4.1, 4.2, 4.3 and 4.4 show the average value, maximum value and variance value of the variances of all the clusters in each codebook obtained from the binary splitting algorithm and the

proposed algorithm using one speaker's speech data. From the result it can be

	no. of clusters	cluster variance		
		ave.	max.	variance
binary splitting	128	3.38	7.79	1.3505
proposed algorithm	128	3.49	5.35	0.6391
proposed algorithm	144	3.31	5.60	0.6207

Table 4.1: Comparison Between Binary Splitting Algorithm and the Nonbinary Algorithm for Cepstral Codebook

	no. of clusters	cluster variance		
		ave.	max.	variance
binary splitting	128	0.20	0.44	0.0049
proposed algorithm	128	0.19	0.26	0.0032
proposed algorithm	144	0.18	0.27	0.0027

Table 4.2: Comparison Between Binary Splitting Algorithm and the Nonbinary Algorithm for Delta-cepstral Codebook

	no. of clusters	cluster variance		
		ave.	max.	variance
binary splitting	128	0.14	0.25	0.0023
proposed algorithm	128	0.14	0.23	0.0011
proposed algorithm	144	0.13	0.18	0.0009

Table 4.3: Comparison Between Binary Splitting Algorithm and the Nonbinary Algorithm for Delta-delta-cepstral Codebook

seen that the proposed algorithm achieves much better partition than binary splitting algorithm according to the optimal k-means partition criterion, i.e., variance of the clusters are more uniform.

	no. of clusters	cluster variance		
		ave.	max.	variance
binary splitting	32	0.50	1.56	0.1409
proposed algorithm	32	0.49	1.15	0.0694
proposed algorithm	36	0.45	1.42	0.0637

Table 4.4: Comparison Between Binary Splitting Algorithm and the Nonbinary Algorithm for Energy Codebook

4.5 The MLP Neural Network and Its Training

As indicated in Figure 4.1, the final stage of the speech recognition systems is the MLP neural network stage.

The inputs to the MLP are obtained from the four codebooks, i.e., cepstral codebook (size 144), delta-cepstral codebook (size 144), delta-delta-cepstral codebook (size 144), and energy codebook (size 36).

Therefore the number of neurons in the input layer of MLP neural network, is equal to the sum of the sizes of four codebooks, i.e., $144 + 144 + 144 + 36 = 468$. The input vector to the MLP, $\mathbf{I}_p$, is defined as a vector of $n_i = 468$ binary elements. Each element corresponds to a unique code in one of the four codebooks. An input element is 1 if the corresponding code is activated by the speech signal in any frame of an utterance, and is 0 if the corresponding code is never activated. In this way the entire waveform of one speech utterance is encoded into a binary vector which serves as the inputs of the MLP neural network.

The size of output layer of the MLP is equal to the number of words to be recognized in the vocabulary, e.g., 20 for the TI20 word data base. The desired output, $\mathbf{d}_p$, is also a binary vector, e.g., with $n_o = 20$ elements. An element in the desired output vector is 1 if the element represents the correct word corresponding to input

vector $\mathbf{I}_p$. All the other output elements should be 0.

The total number of layers of the MLP and the size of the hidden layers are dependent on the application problem which will be discussed in individual examples.

In this approach, the inter-speaker and intra-speaker cepstral variations are directly addressed by the generalization capability of the neural network. The use of the MLP input vectors as defined, in which the codebook codes accumulate over the entire speech utterance onto the MLP input vector, helps to address temporal variations in speech signals. The use of high-order cepstral derivatives partially helps in using sequential information in speech signals.

The following shows a typical result of the MLP input vector which consists of the contributions from four different codebooks (utterance 1 of "one" from female speaker 1 of TI20 data). For convenient presentation, the codes are arranged into a matrix form. The final binary input vector is formulated by concatenating all the rows of these matrices. It is clear that this input vector is very sparse since most of the elements are zero. The actual number of zero and nonzeros in this example are 385 and 83, respectively. The sparse factor is $385/468 = 82\%$.

0	0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	1	0	0	0	0	1	1	0
1	0	0	0	0	0	1	0	1	0	0	0
0	0	0	1	0	1	0	0	0	1	0	0
0	1	0	0	0	0	0	1	1	0	0	0
0	0	0	0	1	0	1	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

output from cepstral codebook


```

0 0 0 1 0 0 1 0 0 0 0 0
0 0 0 1 0 1 0 0 1 1 0 1
0 0 0 1 1 0 0 1 0 0 1 0
0 0 0 1 0 0 0 0 1 0 0 1
0 0 0 0 0 0 1 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 1 0 0
1 0 0 0 1 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0

```

output from delta-cepstral codebook

```

0 0 0 0 1 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0 0 1 1 0
1 0 0 0 0 1 0 0 0 0 0 1
0 0 0 0 0 1 0 0 0 0 1 1
1 0 0 0 0 0 0 1 0 0 1 1
0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0

```

output from delta-delta-cepstral codebook

```

1 0 1 1 0 0
1 0 1 1 1 1
1 0 1 0 0 1
0 0 0 1 0 1
1 0 0 0 0 0
0 0 0 0 0 0

```

output from energy codebook

In the training and testing of our speech recognition MLP, Texas Instrument speech data base TI20 and TI46 were used. The TI20 data base has vocabulary size

of 20, which are "one", "two", "three", "four", "five", "six", "seven", "eight", "nine", "zero", "stop", "start", "rubout", "help", "yes", "no", "repeat", "go", "enter" and "erase", i.e. 10 digits and 10 command words. The T146 data base has a vocabulary size of 46 including the 20 words from T120, and 26 alphabets. The speakers are from different regions in North America. For each word, there are 26 utterances per speaker, of which 10 utterances are from one session and the other 16 utterances are from 8 separate data collection sessions with 2 per session. The first 10 utterances were used as training data and the other 16 utterances were used as testing data. Since for the testing data, the speakers spoke in several different sessions, intra-speaker variations will be even more apparent. Also the testing words were spoken in completely separate sessions from the training words. The use of such testing data will indeed challenge the quality of the neural network speech recognizer. All the words were spoken in isolated utterances.

4.6 Examples by Sparse Training Approach

In this thesis, three cases are presented, speaker dependent with 20 word vocabulary, multi-speaker dependent with 20 word vocabulary and multi-speaker dependent with 46 word vocabulary. In all the cases comparisons among three algorithms are presented, i.e. standard EBP algorithm referred to as EBP, the proposed sparse Algorithm 1 referred to as SEBP1 and the proposed sparse Algorithm 2 referred to as SEBP2.

4.6.1 Example 1

In this example, the training and recognition result from a female speaker is presented. The testing data from the speaker was used to test the MLP neural

network trained by the same speaker's training data, so called speaker dependent. The total number of training patterns is $20\text{words} * 10\text{utterances} = 200$ and the total number of testing patterns is $20\text{words} * 16\text{utterances} = 320$. Since this is an easier task, a two layer MLP neural network is used with 468 inputs and 20 outputs.

In order to identify the potential of the sparse algorithm, we examine the sparsity in the neural network. In Section 4.5, we have presented a specific example of the input binary vector consisting of four codebooks, which was obtained from a particular speech utterance. Here we examine the sparsity of all the 200 utterances involved in training. The result is presented in Figure 4.2 which shows the distribution of the sparse factor among all the training patterns. The sparse factor is computed according to Equation (3.43). The sparsity in general is around 85%.

Three training algorithms, i.e. EBP, SEBP1 and SEBP2, were all used to train the MLP. For the sparse algorithms, the sparse threshold θ^2 was reduced from initial value 0.1 to 1×10^{-6} , the sparse refreshment intervals were adaptively controlled. The size of refreshment intervals in actual training were found to range from 1 to 5. The learning rate η was adaptively controlled according to the parameters: $\eta_{\text{initial}} = 0.2$, $R_{\text{dec}} = 0.7$ and $R_{\text{inc}} = 1.05$.

Figure 4.3 shows the learning curves of the three algorithms, i.e. error E vs. epochs. At beginning, the error reduction in SEBP1 and SEBP2 algorithms is slow. But later in the training, the error reduced down to similar levels as the EBP algorithm. Another interesting phenomenon is that the learning curve of SEBP2 algorithm shows some oscillation. This is because of the refreshment of sparse patterns.

Table 4.5 gives the comparison among the three algorithms, i.e. EBP, SEBP1 and SEBP2. The recognition accuracy from the sparse training is equal to that from the dense training. Compared to EBP, the sparse algorithms SEBP1 (or SEBP2) used

9% (or 14%) more epochs, but the speed per epoch is 5.4 (or 6) times faster. For the overall training, SEBP1 and SEBP2 are about 4.9 and 5.2 times faster than EBP, respectively.

Between the two sparse algorithms, SEBP2 used slightly more epochs than SEBP1, but the speed per epoch in SEBP2 is very fast, resulting in an overall training speed of SEBP2 faster than SEBP1.

Table 4.5: Training and Recognition Results for the TI 20 Data (speaker dependent)

2 layer	CPU time(s)	no. of epochs	training error	testing accuracy
EBP	783	401	2.89×10^{-9}	100%
SEBP1	159	436	2.98×10^{-9}	100%
SEBP2	150	457	2.95×10^{-9}	100%

4.6.2 Example 2

This example is a multispeaker dependent recognition task. The training and recognition results from all eight female speakers are presented. Testing data from all the eight female speakers was used to test the MLP neural network trained by the training data collected from the same eight speakers. The total number of training patterns is $20\text{words} * 10\text{utterances} * 8\text{speakers} = 1600$ and the total number of testing patterns is $20\text{words} * 16\text{utterances} * 8\text{speakers} = 2560$. Since this is a more difficult task, we used two different MLP structures, one being a two layer MLP neural network with 468 inputs and 20 outputs, and the other being a three layer MLP neural network with 468 inputs, 20 outputs and 65 hidden neurons.

Figure 4.4 shows the distribution of the sparse factors among all the training patterns. The sparsity is around 80%.

Three training algorithms, i.e., EBP, SEBP1 and SEBP2, were all used to train the MLP. For the sparse algorithms, the sparse threshold θ^3 was reduced from initial value 0.1 to 1×10^{-4} and θ^2 was kept at 0.1. The sparse pattern refreshment interval was adaptively controlled. In the first 50 epochs, the refreshment was done in every epoch. Towards the end of the training, the refreshment interval were adaptively increased to 5 epochs, i.e., the sparse patterns were fixed for 5 epochs before being updated.

The learning curves of the three algorithms for the 2 layer network are shown in Figure 4.5. The learning curves for the 3 layer network are shown in Figure 4.6. At beginning of training, the error reduction is slow for the SEBP1 and SEBP2 methods. A random perturbation occurs in such situations which is indicated by the sudden jump of the error curve. Later SEBP1 and SEBP2 algorithms converged to similar levels as the EBP algorithm.

Table 4.6 gives the comparison among the three algorithms, i.e. EBP, SEBP1, SEBP2. The recognition accuracy trained by the sparse methods are similar or better than that of the dense EBP. Compared to EBP in the 2 layer case, the sparse algorithms SEBP1 (or SEBP2) used 18% (or 20%) more epochs, but the speed per epoch is 11 (or 17) times faster. For the overall training, SEBP1 and SEBP2 are 9.3 and 14.3 times faster than EBP, respectively.

Compared to EBP in the 3 layer case, the sparse algorithms SEBP1 and SEBP2 used 9% (or 12%) more epochs, but the speed per epoch is about 7.6 (or 8.7) times faster. For the overall training, SEBP1 and SEBP2 are 7 and 7.8 times faster than EBP, respectively.

Between the two sparse algorithms, SEBP2 used slightly more epochs than SEBP1, but the speed per epoch in SEBP2 is very fast, resulting in an overall training speed

of SEBP2 faster than SEBP1.

Table 4.6: Training and Recognition Results for the TI 20 Data (multi-speaker dependent)

2 layer	CPU time(s)	no. of epochs	training error	testing accuracy
EBP (2 layer)	5,627	368	9.8×10^{-8}	99.37%
SEBP1 (2 layer)	604	436	9.1×10^{-8}	99.45%
SEBP2 (2 layer)	395	443	9.4×10^{-8}	99.41%
EBP (3 layer)	10,058	193	9.8×10^{-4}	97.65 %
SEBP1 (3 layer)	1,427	210	9.9×10^{-4}	98.67%
SEBP2 (3 layer)	1,291	217	9.7×10^{-4}	98.86 %

For the three layer network, it is also interesting to look inside the network to see the sparsity of hidden layers. Figures 4.7, 4.8, 4.9 and 4.10 show the histograms of sparse factors ρ_1^2 and ρ_0^2 before and after training. It can be seen that after training the sparsity is increased meaning that speed up is even more significant at the end of training.

4.6.3 Example 3

This is a multi-speaker dependent example similar to Example 2 but with a larger vocabulary. Another 26 alphabets were added and the whole vocabulary includes 46 words. We used the data from all the eight female speakers. The number of training patterns is $46\text{words} * 10\text{utterances} * 8\text{speakers} = 3680$ and the number of testing patterns is $46\text{words} * 16\text{utterances} * 8\text{speakers} = 5888$. Since this is an even more difficult task, a three layer MLP neural network with 468 inputs, 46 outputs and 100 hidden neurons is used.

Figure 4.11 shows the distribution of the sparse factor among all the training

patterns. The sparsity is around 85%. The learning curves of three algorithms are shown in Figure 4.12.

Table 4.7 gives the comparison among the three algorithms, i.e. EBP, SEBP1 and SEBP2. It can be seen that SEBP1 and SEBP2 algorithms give much better testing accuracy in this example. The speed-up ratio of the sparse approaches is around $3 \sim 4$ times over the dense training.

Table 4.7: Training and Recognition Results for the TI 46 Data (multi-speaker dependent)

3 layer	CPU time(s)	no. of epochs	training error	validation accuracy	testing accuracy
EBP	41,318	205	7.17	93.09 %	93.07%
SEBP1	12,624	200	7.16	94.72 %	94.77 %
SEBP2	11,775	248	7.11	94.86 %	94.73 %

Figures 4.13, 4.14, 4.15 and 4.16 show the histograms of sparse factors ρ_1^2 and ρ_0^2 before and after training. It is interesting to notice that the peak of the histogram was split to two peaks after training and the average sparsity increased.

4.7 Conclusion

In this chapter we studied the application of the proposed sparse MLP training technique to speech recognition. A multi-codebook/MLP speech recognition system has been developed for this application and an improved k-means clustering algorithm has been developed and used. Excellent recognition accuracy comparable to that in the literature has been achieved from this system.

The sparse patterns in such applications have been examined. The proposed sparse training algorithms can train the MLP to a very satisfactory result, with testing word accuracy as high as or even higher than that from dense EBP training. The CPU per epoch in the sparse training approach is found to be much less than that in the dense training technique. The total training time for the proposed sparse training technique is shorter than that of the dense EBP for all the examples tested, with SEBP2 on average as the best choice of the training methods used.

Figure 4.1: Diagram of combined multi-codebook/MLP neural network speech recognition system.

Figure 4.2: Distribution of sparse factor ρ_0^1 for training patterns in the speaker dependent case with 20 words vocabulary.

Figure 4.3: Learning curves of EBP algorithm('-.'), SEBP1 algorithm('- -') and SEBP2 algorithm('· ·') for the speaker dependent case with 20 words vocabulary.

Figure 4.4: Distribution of sparse factors ρ_0^1 for training patterns in the multi-speaker dependent case with 20 words vocabulary.

Figure 4.5: Learning curves of EBP algorithm('-.'), SEBP1 algorithm('- -') and SEBP2 algorithm('-') for the multi-speaker dependent case with 20 words vocabulary (2 layer network).

Figure 4.6: Learning curves of EBP algorithm('-.'), SEBP1 algorithm('- -') and SEBP2 algorithm('-') for the multi-speaker dependent case with 20 words vocabulary (3 layer network).

Figure 4.7: Histogram of sparse factor ρ_1^2 before training for multi-speaker dependent case with 20 words vocabulary.

Figure 4.8: Histogram of sparse factor ρ_1^2 after training for multi-speaker dependent case with 20 words vocabulary.

Figure 4.9: Histogram of sparse factor ρ_0^2 before training for multi-speaker dependent case with 20 words vocabulary.

Figure 4.10: Histogram of sparse factor ρ_0^2 after training for multi-speaker dependent case with 20 words vocabulary.

Figure 4.11: Distribution of sparse factors ρ_0^1 for training patterns in the multi-speaker dependent case with 46 words vocabulary.

Figure 4.12: Learning curves of EBP algorithm('-.'), SEBP1 algorithm('- -') and SEBP2 algorithm('-.') for the multi-speaker dependent case with 46 words vocabulary.

Figure 4.13: Histogram of sparse factor ρ_1^2 before training for multi-speaker dependent case with 46 words vocabulary.

Figure 4.14: Histogram of sparse factor ρ_1^2 after training for multi-speaker dependent case with 46 words vocabulary.

Figure 4.15: Histogram of sparse factor ρ_0^2 before training for multi-speaker dependent case with 46 words vocabulary.

Figure 4.16: Histogram of sparse factor ρ_0^2 after training for multi-speaker dependent case with 46 words vocabulary.

Chapter 5

Application to Circuit Extraction

This chapter describes the second type of application for the sparse training technique, i.e., circuit extraction, which is a new application problem for neural networks developed in this work. A new formulation to represent the circuit extraction problem by numerical pattern recognition is proposed. Multilayer perceptrons are utilized to learn the circuit information and to use this knowledge to extract the circuit macro-models. The sparse features in this neural network problem and the experiments with various testing examples are presented. The efficiency of the proposed sparse training technique in training the resultant MLP is demonstrated.

5.1 Introduction

The design of a VLSI circuit is a very time consuming and expensive process. Shortened time to market is critical to the success of a new product. Another task that takes many man-hours is the redesign of existing VLSI circuits whose original documentation or original data base is either missing, inaccurate, or not accessible. Improvement to and reuse of a previous design requires adequate and sometimes very detailed knowledge of the original design. Reverse-engineering is the taking apart of

an existing circuit to find how it is put together. This task is technically much harder than the original design of a circuit.

One of the essential steps in such reverse-engineering procedure is called circuit extraction, which is to extract the high level components from low level descriptions of the circuit. (This concept is different from another concept which is sometimes also called circuit extraction which is to extract circuit from layout.) A specific step considered in this thesis is the extraction of gate level circuit components from transistor level descriptions of the circuit. The circuit descriptions are usually not in schematic form, instead they are in netlist form.

The present approaches to such circuit extraction problem is either through manual extraction or by Rule based automatic circuit extractors [85]. In the case of manual extraction, a schematic diagram is often drawn from the netlist, and an experienced engineer will recognize the high level circuit component based on his knowledge and skill. This may be very labour intensive, especially for large circuits.

In the case of rule-based automatic extraction, an expert system is first developed incorporating all possible rules of circuit connections [86]. The expert system is used to check the given circuit netlist against the rules, and make judgements on the high level components recognized. However in this approach the rules can become very complicated if the size of the high level component becomes large. The number and variety of rules grow rapidly with the size of the circuit and the rules become extremely difficult, or even not feasible to implement.

In this chapter, we develop a new and alternative approach to the circuit extraction problem, i.e., neural network approach. We formulate the circuit extraction problem into a pattern recognition problem. The neural networks will learn various patterns from a large amount of existing circuit information. This knowledge is then used

to perform pattern recognition in the testing circuits, providing the solution of extracted high-level components.

5.2 Problem Description

Circuit extraction is one of the crucial, and also one of the most technically difficult modules, of an overall reverse engineering process. The input information to the circuit extractor module is typically a netlist description of the circuit at the lowest level, i.e., the transistor level. The desired output of the circuit extractor is a high level description of the circuit that is given. The overall extraction for digital circuits is preferably done in a hierarchical manner. Starting from the lowest level, i.e., the transistor level, it will extract the next level components, which is the gate level logic components, e.g., NOT, AND, OR, NAND, NOR, XOR. Theoretically the extraction process should continue to higher levels, and eventually to system level components. Notice that while several lower level components make one higher level component, the sequence of the lower level components in the netlist may vary, complicating the extraction process. In reality, automatic extraction of these higher level (higher than gate level) components still remains an unsolved problem. The lack of an efficient mathematical formulation of the extraction problem is one of the gaps that needs to be filled.

In this chapter, we derive an alternative formulation of the circuit extraction problem using the concept of neural network pattern recognition. Within the scope of this thesis work, we aim at the transistor to gate level extraction problem to show the feasibility of the neural network method and to demonstrate the application of the proposed sparse EBP training algorithms.

5.3 MLP Formulation for Circuit Extraction

A formulation using the binary vector to represent the topology of the circuit is derived in this section. The input information to our extraction task is a netlist. The following is an example of a portion of a CMOS circuit netlist.

```

P  B  C  VDD
P  A  D  C
N  B  D  GND
N  A  D  GND

```

Now the purpose is to recognize what type of gate this portion of netlist represents. In this netlist, each line represents a lower level component, e.g., transistor in this example. The first symbol of a line represents the type of component, followed by the nodes of the component. In this example, component type symbols P and N are for p-type and n-type transistors, respectively. The three nodes in the netlist following P or N are the gate, drain and source nodes, respectively. Symbols A, B, C and D are nodes in the circuit. VDD and GND represent the power supply node and the ground node, respectively.

A manual solution usually starts with drawing (or imagining) a schematic representation from the netlist, and from which to recognize what gate this is. A rule based approach will check the netlist against various rules. In our approach, we need to represent this netlist text description into numerical patterns acceptable by a neural network, and at the same time, no necessary information in the netlist is lost.

The solution of the extracted high level component for this example should be the logic component NOR, with A and B as inputs and D as the output, i.e., the netlist

at the gate level should be

$$NOR \quad A \quad B \quad D$$

where, again the first symbol represents the type of the component (i.e., NOR), followed by the nodes of the component (i.e., A, B, D).

Now the specific task is how to represent the netlist at the transistor level as a feature vector for neural network input, and the netlist at the gate level as another feature vector for neural network output.

To facilitate the description, we use the word "component" to represent lower level components. The word "macro" or "macro-model" is used to represent the high level components. For the example given above, components mean the transistors. Macros mean the NOR gate. The extraction problem is to extract the macros from a given netlist of the components.

Let N_{com} be the number of lower level components that are simultaneously checked during the extraction process. Let N_{node} be the maximum number of nodes per component. Each type of components is identified by a binary word, with N_{bit} as the number of bits in such a word. Let N_{mac} be the number of library macro-models that are gathered as candidates for the solution of extracted types of macro-models.

We define a component to component incidence matrix C , containing N_{com} by N_{com} blocks. Each block entry of the matrix, i.e., block c_{ij} , represents the connection between two components, i.e., between component i and component j . We further define that the connection between the two components is represented by a node to node incidence matrix, containing N_{node} by N_{node} entries. For example, c_{ij} is represented by a N_{node} by N_{node} matrix whose (k, l) th entry represent the connection between node k of component i and node l of component j . Therefore the final

connection matrix **C** is a 4-dimensional matrix of $\mathbf{C}[N_{com}][N_{com}][N_{node}][N_{node}]$ defined as,

$$\mathbf{C} = [c_{ijkl}]_{N_{com} \times N_{com} \times N_{node} \times N_{node}} \quad (5.1)$$

$$c_{ijkl} = \begin{cases} 1 & \text{if node } k \text{ of component } i \text{ is} \\ & \text{connected to node } l \text{ of component } j \\ 0 & \text{otherwise} \end{cases} \quad (5.2)$$

$$i, j = 1, 2, \dots, N_{com},$$

$$k, l = 1, 2, \dots, N_{node}$$

Another part for the input of the neural network is the component type Matrix **T**, which is a 2-dimension matrix. $\mathbf{T}[N_{com}][N_{bit}]$, defined as

$$\mathbf{T} = [t_1 \quad t_2 \quad \dots, \quad t_{N_{com}}] \quad (5.3)$$

where t_i is a vector of N_{bit} binary entries representing the component type of component i .

Figure 5.1 shows the connection matrix **C** and component type matrix **T** for a general extraction problem. These two matrices form the input feature vector of the MLP.

For the definition of the outputs of the MLP, we define output macro selection vector **D**, and the macro node selection matrix **H**. Vector **D** is defined as

$$\mathbf{D} = [d_i]_{N_{mac}} \quad (5.4)$$

$$d_i = \begin{cases} 1 & \text{if macro } i \text{ is the correct macro for the input matrices } \mathbf{C} \text{ and } \mathbf{T} \\ 0 & \text{otherwise} \end{cases} \quad (5.5)$$

2 of /de 2

PM-1 3 1/2" x 4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

Neural Network input.

Connection Matrix C :

		component 1	component i	component N_{com}
		node 1 ... node N_{node}	node k	node 1 ... node N_{node}
component 1	node 1	0 * ... * 0	...	0 * ... * 0
	node N_{node}	0 * ... * 0	...	0 * ... * 0
component j	node l	...	c_{jkl}	...
component N_{com}	node 1	0 * ... * 0	...	0 * ... * 0
	node N_{node}	0 * ... * 0	...	0 * ... * 0

Type Matrix T :

component 1	component i	component N_{com}
bit 1 ... bit N_{bit}	bit j	bit 1 ... bit N_{bit}
0 ... 0	t_{ij}	0 ... 1

Figure 5.1: The format of the connection matrix C and component type matrix T for the extraction problem. These two matrices form the input feature vector of the MLP. Each numerical entry in the matrices is either 1 or 0. The values in this figure are for illustration purpose only.

$$i = 1, 2, \dots, N_{mac}$$

Macro node selection matrix **H** can be defined as

$$\mathbf{H} = [h_{i,j,k}]_{2 \times N_{com} \times N_{node}} \quad (5.6)$$

$$h_{1,j,k} = \begin{cases} 1 & \text{if node } k \text{ of component } j \text{ is an input node of the macro} \\ 0 & \text{otherwise} \end{cases} \quad (5.7)$$

$$h_{2,j,k} = \begin{cases} 1 & \text{if node } k \text{ of component } j \text{ is an output node of the macro} \\ 0 & \text{otherwise} \end{cases} \quad (5.8)$$

Figure 5.2 shows the vector **D** and matrix **H**, for a general situation. They represent the output layer of the MLP neural network circuit extractor.

The actual extraction process is done through several passes, each pass extracting a subclass of macros. Notice that we separated the extraction process into two cases, i.e., partial model extraction and full model extraction. When both **D** and **H** are used as outputs in the MLP, they represent the complete information of the macro-model, including type of the macro and its nodes. During training, the neural network has to be optimized to match the type of the macro and its nodes in the training data. This would be meaningful only for the correct macros. If the input information, i.e., **C** and **T** matrices, do not belong to any macros supplied for this extraction pass, all entries of the **D** vector should be zero. However in this case, the **H** matrix is not definable, and if we force the **H** matrix to be all zero or to be any pattern during training, the neural network would be forced to learn unnecessary relationships. The solution to this problem is to divide the recognition into partial model and then full model cases. The extraction starts with partial model extraction in which the only purpose is to extract the type of macros, without finding the macro nodes. In this case, the extractor will likely reject many illegal patterns (i.e., not belonging to the

Neural Network output:

Partial Model case:

macro selection vector **D**:

macro 1	macro i	macro N_{mac}
1	d_i	0

Full Model case:

macro node selection matrix **H**:

	component 1		component j	component N _{com}			
	node 1	node N _{node}	node k	node 1	node N _{node}		
macro input node	0	...	0	h_{1jk}	1	...	0
macro output node	0	...	0	h_{2jk}	0	...	0

macro selection vector **D**:

macro 1	macro i	macro N_{mac}
1	d_i	0

Figure 5.2: The format of outputs of the MLP circuit extractor. Each numerical entry in **D** and **H** is either 1 or 0. The values in this figure are for illustration purpose only.

given subclass of macros), and find the few legal patterns that belong to the subclass of macros. Then the full model extraction is applied to only the legal patterns to extract the correct nodes for the macros in the given subclass.

In summary, the number of inputs to the neural network, n_i , is

$$n_i = N_{com}^2 \times N_{node}^2 + N_{com} \times N_{bit} \quad (5.9)$$

While the number of outputs of neural network, n_o , is

$$n_o = N_{mac} \quad (5.10)$$

for the partial model case, or

$$n_o = N_{mac} + 2 \times N_{com} \times N_{node} \quad (5.11)$$

for the full model case.

5.4 An Example of MLP Inputs and Outputs

In this section we use an example to further illustrate the MLP input and output definitions for the circuit extraction problem. A simple CMOS NOT gate (i.e., inverter) example is used, through which the entire component connection matrix **C**, component type matrix **T**, macro selection vector **D** and macro node selection matrix **H** are defined. The netlist at the transistor level is:

P A Y VDD

N A Y GND

where P and N represent p-type and n-type transistors, respectively. The 3 nodes following P or N are gate, drain and source nodes, respectively. A and Y are nodes in

the circuit. VDD and GND represent the power supply node and the ground node, respectively.

At the high level, the correctly extracted macro should be a NOT gate, and in netlist form, should be

$$\text{NOT} \quad A \quad Y$$

where again, the first symbol represents the type of the macro, i.e., NOT, followed by the nodes. In this example, the symbols A and Y represent the input and the output nodes of the NOT gate.

We also define a pseudo-component, called "Power-Ground" component which, for the CMOS circuit example presented above, has two nodes, VDD and GND. Therefore, to form the connection matrix C for the gate NOT, we use a total of three basic components, i.e., power-ground component, p-type transistor, and n-type transistor. In each transistor there are three nodes, i.e., gate node (G), drain node (D) and source node (S). Although the power-ground component has only two nodes, i.e., power node (VDD) and ground node (GND), a third dummy node(*) is added to make the connection matrix more uniformly structured.

The complete topological relationship of the circuit components are defined by the connection matrix C , which is a type of incidence matrix representing component to component incidence matrix, and more specifically, component node to component node incidence matrix. The size of C is $3 \times 3 \times 3 \times 3$. The sequence of components in the matrix is the power ground component followed by other components in the same order as that in the netlist. The sequence of nodes inside the components are (G D S) or (VDD GND *) which is also the same as the sequence in the netlist. If $C = 0$, it means that all the components are disconnected. When any entry is set to

"1", the corresponding two nodes are connected. For example, the drain node of p type transistor is connected to the drain node of n type transistor, so $c_{2322} = 1$ and $c_{3222} = 1$. Matrix **C** is symmetric. For the component type matrix **T**, suppose the component types are represented by a 3 bit binary word, then the dimension of **T** is 3 by 3. The sequence of the codes for the components is the same as that in connection matrix **C**. Finally the total number of inputs for neural network, i.e., the number of entries in **C** and **T**, is 90. Figure 5.3 shows the complete neural network input vector for gate NOT.

For neural network outputs, there are two cases, i.e., partial model case and full model case, both being binary type outputs. Figure 5.4 shows the macro selection vector **D** and the macro node selection matrix **H** for gate NOT. In the partial model case, the output is similar to that in speech recognition, i.e., competitive output. Only one output is "1" representing the correct macro-model while all others should be "0". The number of outputs equals to the number of macro models that are legal in this extraction process. In this example, the legal macros (or the subclass of macros) for this extraction pass are NOT and TGATE (transmission gate). In the full model case, the outputs contain an extra matrix **H** to represent the input and output nodes of the macro NOT. The first dimension of **H** is 2, representing input and output, respectively. In the input row, if any column is "1", the corresponding node is a macro input node. Similarly in the output row, if any entry is "1", the corresponding node is a macro output node. Figure 5.4 shows the complete neural network output vector for gate NOT.

Now consider the sparsity of the MLP input pattern, which consists of connection matrix **C** and type matrix **T**. For the NOT gate example presented in this section, the number of non-zeros in the MLP input vector is 10, and the number of zeros is

Neural Network input:

Connection Matrix C:

	Power-ground			p transistor			n transistor		
	VDD	GND	*	G	D	S	G	D	S
VDD	0	0	0	0	0	1	0	0	0
GND	0	0	0	0	0	0	0	0	1
*	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	1	0	0
D	0	0	0	0	0	0	0	1	0
S	1	0	0	0	0	0	0	0	0
G	0	0	0	1	0	0	0	0	0
D	0	0	0	0	1	0	0	0	0
S	0	1	0	0	0	0	0	0	0

Type Matrix T:

Power-ground			p transistor			n transistor		
0	0	0	0	1	0	0	0	1

Figure 5.3: MLP input parameters for the gate NOT (inverter).

Neural Network output:**Partial Model case:**

macro selection vector D:

NOT	TGATE
1	0

Full Model case:

macro node selection matrix H:

	Power-ground			p transistor			n transistor		
	VDD	GND	*	G	D	S	G	D	S
macro input node	0	0	0	1	0	0	1	0	0
macro output node	0	0	0	0	1	0	0	1	0

macro selection vector D:

NOT	TGATE
1	0

Figure 5.4: MLP output parameters for the gate NOT (inverter).

80. The sparse factor ρ_0^1 is $80/90 = 90\%$.

5.5 Neural Network Training by the Sparse Approach

In this thesis, two examples of circuit extraction are presented. One is to extract gate NOT and gate TGATE (transmission gate). The other one is to extract gate NAND and gate NOR. Several circuits from [87], [88], [89] are used to train and test the neural network. These circuits are parallel register (with 54 transistors), D flipflop (with 28 transistors), multiplexer (with 190 transistors), decoder (with 108 transistors), encoder (with 172 transistors) and full adder (with 62 transistors). The netlist of parallel register and D-Flipflop was written by one person while the netlists of other circuits were written by another person. These two sets of data are completely independent from each other, and can be used as training patterns and testing patterns respectively. According to the results from speech recognition, SEBP2 algorithm gives the best overall performance. In the circuit extraction application, the comparison of the sparse MLP training algorithm with dense EBP is done between SEBP2 and EBP.

5.5.1 Example 1

In this example, gate NOT and gate TGATE are extracted from the various circuits. Table 5.1 shows the number of patterns in each circuit together with the number of inputs and number of outputs for the MLP. In the full model case, the number of patterns is much less than that in the partial model case because the partial model case includes many wrong patterns (illegal patterns) that are present in the circuit netlist and belong to neither NOT nor TGATE. However the presence of these

patterns in training the partial model MLP is essential in improving the discriminating ability of neural network, i.e., to distinguish between legal (NOT, TGATE) patterns and illegal ones. Here we used parallel register and an additional D-Flipflop as the training circuit and used the multiplexer, decoder, encoder and full-adder circuits as the testing circuits. So in the full model case, there are 9 training patterns and 53 testing patterns and in the partial model case, there are 78 training patterns and 476 testing patterns.

Table 5.1: NOT-TGATE Case: List of Number of MLP Inputs, Outputs and Patterns

circuit	full model			partial model		
	inputs	outputs	patterns	inputs	outputs	patterns
register and a D-Flipflop	153	26	9	153	2	78
multiplexer	153	26	17	153	2	166
decoder	153	26	7	153	2	84
encoder	153	26	25	153	2	168
full adder	153	26	4	153	2	58

Since there are not too many combinations of transistors in the NOT and TGATE case, a 2 layer network is enough for the task. Also the sparsity of the inputs is nearly identical across different training patterns because of its dependence on the connected nodes of transistors. For full model extraction, the 2 layer MLP neural network has 153 inputs and 26 outputs and for the partial model case the 2 layer MLP neural network has 153 inputs and 2 outputs.

The neural network was trained by both the dense EBP method and the proposed sparse training algorithm SEBP2. Figures 5.5 and 5.6 show the learning curves of EBP and SEBP2 algorithms for full model and partial model cases, respectively.

Table 5.2 gives the comparison between algorithms EBP and SEBP2. It can be

seen that both algorithms achieved the same extraction accuracy of 100 % for the testing data when reaching similar training error level. SEBP2 took about 30% more epochs in both the full and partial model cases. But the time used per epoch by SEBP2 is much less than that of EBP, with the ratio being about 1/2.5 in the full model case and 1/4.2 in the partial model case. A computational speedup for the overall training is obtained by using the sparse training algorithm SEBP2.

Table 5.2: NOT-TGATE Case: Comparion of EBP and Sparse Training Approach SEBP2

	full model		partial model	
	dense(EBP)	sparse(SEBP2)	dense(EBP)	sparse(SEBP2)
no. of layers	2	2	2	2
CPU time(s)	2.67	1.42	3.07	0.97
no. of epochs	44	58	80	105
training error	0.096	0.099	0.097	0.099
testing accuracy	100%	100%	100%	100%

5.5.2 Example 2

In this example, 2-input gates NAND and NOR are extracted from the various circuits. Table 5.3 shows the number of patterns in each circuit together with number of inputs and number of outputs for MLP. Here we use parallel register and a D-Flipflop as the testing circuits and use the other circuits (i.e. multiplexer, decoder, encoder, full adder) as the training circuits. In the full model case, there are 37 training patterns and 15 testing patterns. In the partial model case, there are 468 training patterns and 76 testing patterns.

A 2 layer neural network is used for the full model extraction case for the gates NAND and NOR. Also the sparsity of the inputs is identical across different training

Table 5.3: NAND-NOR Case: List of Number of MLP Inputs, Outputs and Patterns

circuit	full model			partial model		
	inputs	outputs	patterns	inputs	outputs	patterns
register and a D-Flipflop	415	42	15	415	2	76
multiplexer	415	42	17	415	2	164
decoder	415	42	0	415	2	82
encoder	415	42	9	415	2	166
full adder	415	42	11	415	2	56

natterns because of its dependence on the connected nodes of transistors. This 2 layer MLP neural network has 415 inputs and 42 outputs. In the partial model case, the variety of patterns is large. A 2 layer neural network has been found inadequate for this pattern recognition purpose. The training accuracy cannot reach 100 %. A satisfactory solution is reached using a 3 layer MLP. The neural network has 415 inputs, 2 outputs and 30 hidden neurons. Figure 5.7 shows the distribution of the sparse factors among all the training patterns and figure 5.8 shows the histogram of the sparse factors. The average sparsity is around 95%.

The neural network was trained using both the dense and the sparse training algorithms, i.e., EBP and SEBP2. Figures 5.9 and 5.10 show the learning curves of EBP and SEBP2 algorithms for the full model and the partial model cases, respectively. Some oscillation is observed in the learning curve of partial model case due to sparse pattern refreshment.

Table 5.4 gives the comparison between algorithms EBP and SEBP2. It can be seen that both algorithms achieved the same extraction accuracy of 100 % for the testing data and reached similar training error level at the end of training. It is interesting to notice that in the full model case, EBP and SEBP2 took the same

number of epochs to reach similar training error and 100 % testing accuracy. But the CPU used by SEBP2 is only 13% of that of EBP. The overall speedup factor by the sparse training over dense training for the full model case is 7.3 and for the partial model case is 5.3.

Table 5.4: NAND-NOR Case: Comparison of EBP and Sparse Training Approach SEBP2

	full model		partial model	
	dense(EBP)	sparse(SEBP2)	dense(EBP)	sparse(SEBP2)
no. of layers	2	2	3	3
CPU time(s)	63.49	8.73	4,272	807
no. of epochs	80	80	448	675
training error	1.93	1.94	9.8×10^{-11}	9.3×10^{-11}
testing accuracy	100%	100%	100%	100%

Figures 5.11, 5.12, 5.13 and 5.14 show the histograms of sparse factors ρ_1^2 and ρ_0^2 before and after training. In this case, the peak of the histogram didn't move to high position after training. But the contrast of the peak with the side lobe is increased, especially on the left side. The average sparsity is still increased.

5.6 Conclusion

The second type of application for the sparse training technique, i.e., circuit extraction, which is a new application for neural networks has been presented in this chapter. A set of matrices and vectors have been defined that can represent circuit netlist information, and can be used as neural network inputs and outputs. Multilayer perceptrons are utilized to learn the circuit information and to use this knowledge to extract the circuit macromodels. The sparse features in this neural network problem and the experiments with various testing examples are presented. The proposed

sparse training technique in training the resulting MLP circuit extraction examples has provided much computational speedup over the conventional dense training technique.

Figure 5.5: Learning curves of EBP algorithm('-.') and SEBP2 algorithm('-.') for NOT and TGATE extraction in the full model case.

Figure 5.6: Learning curves of EBP algorithm(‘-’) and SEBP2 algorithm(‘-’) for NOT and TGATE extraction in the partial model case.

Figure 5.7: Distribution of sparse factor ρ_0^1 for training patterns for NAND-NOR extraction in the partial model case.

Figure 5.8: Histogram of input layer sparse factor ρ_0^1 for NAND-NOR extraction in the partial model case.

Figure 5.9: Learning curves of EBP algorithm('-.') and SEBP2 algorithm('-.') for NAND and NOR extraction in the full model case.

Figure 5.10: Learning curves of EBP algorithm('-.') and SEBP2 algorithm('-.') for NAND and NOR extraction in the partial model case.

Figure 5.11: Histogram of sparse factor ρ_1^2 before training for NAND-NOR extraction in the partial model case.

Figure 5.12: Histogram of sparse factor ρ_1^2 after training for NAND-NOR extraction in the partial model case.

Figure 5.13: Histogram of sparse factor ρ_0^2 before training for NAND-NOR extraction in the partial model case.

Figure 5.14: Histogram of sparse factor ρ_0^2 after training for NAND-NOR extraction in the partial model case.

Chapter 6

Conclusions and Further Research

6.1 Conclusions

This thesis was motivated by the potential for opening new application areas of neural networks in electrical design, and by the increasing need for a fast and efficient training algorithm to ensure the success of these new applications. The work is accomplished by successfully bridging several gaps between neural network and electrical design approaches. More specifically, several contributions have been presented in this thesis to address this issue:

1. A new and efficient sparse backpropagation approach has been proposed. This approach is formulated by combining the sparse optimization concept used in circuit optimization areas, with backpropagation method which is the most influential training technique for multilayer perceptrons. Instead of using the conventional sparse concept which is based on problem topology and which does not fit neural network situation, we approach it through an different point of view, i.e., numerical sparsity, which is inherent in the neural network environment. The sparsity in neural network is identified and used to formulate a new training method using a new set of feedforward and backpropagation formulae.

2. A systematic computational algorithm is developed and tested that implements the sparse training approach in an efficient way. A completely new set of parameters that regulate the sparse learning process is introduced. An adaptation procedure for all the new training parameters is proposed. The new sparse training algorithm becomes identical to the standard backpropagation training when the training parameters assume extreme values, e.g., setting the sparse threshold to zero. Proper choice and adaptation of the new training parameters will result in an efficient training procedure much faster than standard backpropagation.

3. The validity and efficiency of the proposed training algorithm are confirmed by two completely different types of applications, i.e., speech recognition and circuit extraction. For speech recognition, an improved clustering algorithm is developed to obtain the codebooks as the input stage for MLP recognition. Excellent recognition results have been achieved using the system on Texas Instruments isolated speech data TI-20 and TI-46. The sparsity in the MLP in the speech recognition system has been exploited. Speedup factors of 2 to 10 have been achieved using the proposed sparse training over the conventional backpropagation.

4. The second type of application is a new application area for neural networks, i.e., circuit extraction. A new formulation to represent the electrical topological extraction problem by numerical pattern recognition has been derived. Electrical rules in the original extraction problem have been translated into numerical representations so that they become neural network input/output parameters. Multilayer perceptrons have been experimented with to learn the circuit information. The neural network then uses this knowledge to extract the circuit macro-models through pattern recognition. This new formulation has been confirmed by extraction examples.

5. The proposed sparse training technique has been implemented into a computer

program for general multilayer perceptron training. The program is written in C++. It allows users to use any number of hidden layers, and any number of neurons in each layer, limited only by the memory space available in computers. All the newly introduced learning parameters and adaptation schemes are implemented and are controllable by the user. This program has been verified by all the speech recognition and circuit extraction examples presented in this thesis. For all the cases, speedup over standard backpropagation has been consistently achieved by using this program.

6.2 Suggestions for Future Research

1. The proposed sparse training technique can be exploited to make the neural network approach for circuit modeling problems more attractive. Modeling the high-speed VLSI interconnect delay is recently done using MLP as a continuous multi-dimensional universal nonlinear approximator. However in many practical cases, the interconnect termination parameters, e.g., resistor values, are discrete rather than continuous. A new formulation converting electrical representation into truly discrete patterns and applying a discrete MLP representation will be a possible new direction. In this case, the proposed sparse training algorithm can be fully exploited to achieve faster training, thus making the neural network approach to interconnect modeling more efficient.

2. The sparse concept is closely related to large-scale problem solving. To further exploit the advantage of sparse training technique, we plan to develop applications where large neural networks are important. For example, in the speech recognition example, one future direction is to use a large vocabulary speech recognition task.

As the size of the vocabulary increases, so does the size of the neural network. The concept of sparse optimization becomes more important in this case. A comparison between conventional backpropagation and the proposed sparse training will reveal potentially even more significant speedup by the new approach.

3. Future direction for the circuit extraction problem is to consider the next level of hierarchical structure, i.e. from extraction of gate level macro-model to the extraction of higher level components. The problem in this case becomes more complicated as the number and variety of macro-models both increase. A one-hundred percent recognition accuracy might become difficult to achieve. Several best possible guesses for extraction results might be produced.

4. Expansion of the sparse optimization approach to other MLP training approaches and other types of neural networks, e.g., recurrent neural networks, is another possible future direction. The existing training of recurrent neural networks is based on real time backpropagation and backpropagation through time. The sparse training concept can be incorporated into such training scheme to provide potential speedup over the existing dense training approach.

Bibliography

- [1] J. W. Bandler and Q. J. Zhang, "Optimization Techniques for Modeling, Diagnosis and Tuning," In T. Ozawa, editor, *Analog Methods for Computer-Aided Circuit Analysis and Diagnosis*, Marcel Dekker, New York, 1988.
- [2] J. W. Bandler and S. H. Chen, "Circuit Optimization: the State of the Art," *IEEE Trans. Microwave Theory and Techniques*, vol. 36, pp. 424-443, 1988 (invited).
- [3] K. K. Low and S. W. Director, "A New Methodology for the Design Centering of IC Fabrication Process," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 895-903, 1991.
- [4] R. L. Liu, Q. J. Zhang, and M. S. Nakhla, "A Frequency-domain Approach to Performance Optimization of High-speed VLSI Interconnects," *IEEE Trans. Microwave Theory and Techniques*, vol. 40, pp. 2403-2411, 1992.
- [5] Q. J. Zhang, S. Lum, and M. S. Nakhla, "Minimization of Delay and Crosstalk in High-speed VLSI Interconnects," *IEEE Trans. Microwave Theory and Techniques*, vol. 40, pp. 1555-1563, 1992.

- [6] Q. J. Zhang and M. S. Nakhla, "Statistical Simulation and Optimization of VLSI Interconnects," *Int. Journal Analog Integrated Circuits and Signal Processing*, vol. 5, pp. 95–106, 1994.
- [7] E. Sanchez-Sinencio and C. Lau, *Artificial Neural Networks: Paradigms, Applications, and Hardware Implementation*, IEEE Press, New York, 1992.
- [8] N. B. Karayiannis and A. N. Venetsanopoulos, *Artificial Neural Networks: Learning Algorithms, Performance Evaluation, and Applications*, Kluwer Academic Publishers, Boston, Massachusetts, 1993.
- [9] S. Haykin, *Neural Networks: A Comprehensive Foundation*, IEEE Press, New York, 1994.
- [10] R. M. M. Chen and W. W. Chan, "An Efficient Tolerance Design Procedure for Yield Maximization Using Optimization Techniques and Neural Network," In *Proc. IEEE Int. Symposium Circuits and Systems*, pp. 1793–1796, Chicago, Illinois, 1993.
- [11] Q. J. Zhang and M. S. Nakhla, "Signal Integrity Analysis and Optimization of VLSI Interconnects Using Neural Network Models," In *Proc. IEEE Int. Symposium Circuits and Systems*, pp. 459–462, London, England, 1994.
- [12] M. Vai and S. Prasad, "Automatic Impedance Matching with a Neural Network," *IEEE Microwave and Guided Wave Letters*, vol. 3(10), pp. 353–354, October 1993.

- [13] Y. Li, R. L. Mahajan, and J. Tong, "Design Factors and Their Effect on PCB Assembly Yield-statistical and Neural Network Predictive Models," *IEEE Trans. Comp., Pack., and Manuf. Technol., Part A*, vol. 17(2), pp. 174-182, 1994.
- [14] A. H. Zaabab, Q. J. Zhang, and M. S. Nakhla, "Analysis and Optimization of Microwave Circuits and Devices Using Neural Network Models," In *IEEE Int. Microwave Symposium Digest*, pp. 393-396, San Diego, California, 1994.
- [15] A. Veluswami, Q. J. Zhang, and M. S. Nakhla, "A Neural Network Model for Propagation Delays in Systems with High Speed VLSI Interconnect Networks," In *Proc. IEEE Custom Integrated Circuits Conference*, pp. 387-390, Santa Clara, California, 1995.
- [16] A. Veluswami, H. Zaabab, Q. J. Zhang, and M. S. Nakhla, "The Application of Neural Networks to High-speed VLSI Interconnect Simulation," In *Proc. Canadian Conf. Electrical and Computer Engineering*, Montreal, Canada, September 1995.
- [17] A. H. Zaabab, Q. J. Zhang, and M. S. Nakhla, "Application of Neural Networks in Circuit Analysis," In *Proc. IEEE Int. Conf. Neural Networks*, Perth, Australia, November 1995 (accepted).
- [18] A. H. Zaabab, Q. J. Zhang, and M. S. Nakhla, "A Neural Network Modeling Approach to Circuit Optimization and Statistical Design," *IEEE Trans. Microwave Theory and Techniques*, vol. 43(6), pp. 1349-1358, June 1995.
- [19] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," In D. E. Rumelhart and J. L. McClelland, editors,

- Parallel Distributed Processing*, volume I, pp. 318–362, MIT Press, Cambridge, Massachusetts, 1986.
- [20] D. G. Luenberger, *Linear and Nonlinear Programming*, Addison-Wesley, Reading, Massachusetts, 1989.
- [21] N. Morgan, “Big Dumb Neural Nets: A Working Brute Force Approach To Speech Recognition,” In *Proc. IEEE Int. Conf. Neural Networks*, volume VII, pp. 4462–4465, Orlando, Florida, July 1994.
- [22] F. Wang and Q. J. Zhang, “A Sparse Matrix Approach to Neural Network Training,” In *Proc. IEEE Int. Conf. Neural Networks*, Perth, Australia, November 1995 (accepted).
- [23] K. Ochiai, N. Toda, and S. Usui, “Kick-out Learning Algorithm to Reduce the Oscillation of Weights,” *Neural Networks*, vol. 7(5), pp. 797–807, 1994.
- [24] S. J. Perantonis and D. A. Karras, “An Efficient Constrained Learning Algorithm with Momentum Acceleration,” *Neural Networks*, vol. 8(2), pp. 237–249, 1995.
- [25] *Neural Network Toolbox: For Use with Matlab*, The MathWorks Inc., Natick, Massachusetts, 1993.
- [26] R. A. Jacobs, “Increased Rate of Convergence Through Learning Rate Adaptation,” *Neural Networks*, vol. 1, pp. 295–307, 1988.
- [27] R. Battiti, “Accelerated Backpropagation Learning: Two Optimization Methods,” *Complex Systems*, vol. 3, pp. 331–342, 1989.

- [28] M. Arisawa and J. Watada, "Enhanced Back-propagation Learning and Its Application to Business Evaluation," In *Proc. IEEE Int. Conf. Neural Networks*, volume I, pp. 155-160, Orlando, Florida, July 1994.
- [29] A. G. Parlos, B. Fernandez, A. F. Atiya, J. Muthusami, and W. K. Tsai, "An Accelerated Learning Algorithm For Multilayer Perceptron Networks," *IEEE Trans. Neural Networks*, vol. 5, pp. 493-497, May 1994.
- [30] X. H. Yu, G. A. Chen, and S. X. Cheng, "Dynamic Learning Rate Optimization of the Backpropagation Algorithm," *IEEE Trans. Neural Networks*, vol. 6(3), pp. 669-677, May 1995.
- [31] N. Baba, Y. Mogami, M. Kohzaki, Y. Shiraishi, and Y. Yoshida, "A Hybrid Algorithm for Finding the Global Minimum of Error Function of Neural Networks and Its Applications," *Neural Networks*, vol. 7(8), pp. 1253-1265, 1994.
- [32] P. P. Van-Der-Smagt, "Minimisation Methods for Training Feedforward Neural Networks," *Neural Networks*, vol. 7(1), pp. 1-11, 1994.
- [33] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, Cambridge, Massachusetts, 1986.
- [34] D. R. Hush and J. M. Salas, "Improving the Learning Rate of Back Propagation with the Gradient Reuse Algorithm," In *Proc. IEEE Int. Conf. Neural Networks*, volume I, pp. 441-447, San Diego, California, 1988.
- [35] D. B. Parker, "Optimal Algorithms for Adaptive Networks: Second Order Back Propagation, Second Order Direct Propagation and Second Order Heb-

- bian Learning," In *Proc. IEEE First Int. Conf. Neural Networks*, volume II, pp. 593–600, San Diego, California, 1987.
- [36] R. L. Watrous, "Learning Algorithms for Connectionist Networks: Applied Gradient Methods of Nonlinear Optimization," In *Proc. IEEE First Int. Conf. Neural Networks*, volume II, pp. 619–627, San Diego, California, 1987.
- [37] L. P. Ricotti, S. Ragazzini, and G. Martinelli, "Learning of Word Stress in a Sub-optimal Second Order Backpropagation Neural Network," In *Proc. IEEE Int. Conf. Neural Networks*, volume I, pp. 355–361, San Diego, California, 1988.
- [38] S. Kollias and D. Anastassiou, "An Adaptive Least Squares Algorithm for the Efficient Training of Artificial Neural Networks," *IEEE Trans. Circuits and Systems*, vol. 36, pp. 1092–1101, 1989.
- [39] M. G. Bello, "Enhanced Training Algorithms, and Integrated Training/Architecture Selection for Multilayer Perceptron Networks," *IEEE Trans. Neural Networks*, vol. 3(6), pp. 864–875, November 1992.
- [40] W. L. Buntine and A. S. Weigend, "Computing Second Derivatives in Feed-Forward Networks: A Review," *IEEE Trans. Neural Networks*, vol. 5(3), pp. 480–488, 1994.
- [41] R. Battiti, "First and Second-Order Methods for Learning: Between Steepest Descent and Newton's Method," *Neural Computation*, vol. 4, pp. 141–166, 1992.
- [42] A. H. Kramer and A. Sangiovanni-Vincentelli, "Efficient Parallel Learning Algorithms for Neural Networks," In D. S. Touretzky, editor, *Advances in Neural*

Information Processing Systems, pp. 40–48, Morgan Kaufmann, San Mateo, California, 1988.

- [43] S. Ergezinger and E. Thomsen, "An Accelerated Learning Algorithm for Multilayer Perceptrons: Optimization Layer by Layer," *IEEE Trans. Neural Networks*, vol. 6, pp. 31–42, January 1995.
- [44] J. S. Shawe-Taylor and D. A. Cohen, "Linear Programming Algorithm for Neural Networks," *Neural Networks*, vol. 3, pp. 575–582, 1990.
- [45] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization By Simulated Annealing," *Science*, vol. 220, pp. 671–680, 1983.
- [46] T. Rönkvallsson, "On Langevin Updating in Multilayer Perceptron," *Neural Computation*, vol. 6, pp. 916–926, 1994.
- [47] R. Brunelli, "Training Neural Nets Through Stochastic Minimization," *Neural Networks*, vol. 7(9), pp. 1405–1412, 1994.
- [48] S. Singhal and L. Wu, "Training Feedforward Networks with the Extended Kalman Algorithm," In *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, pp. 1187–1190, Glasgow, Scotland, 1989.
- [49] S. A. Solla, E. Levin, and M. Fleisher, "Accelerated Learning in Layered Neural Networks," *Complex Systems*, vol. 2, pp. 625–640, 1988.
- [50] R. Reed, "Pruning Algorithms – A Survey," *IEEE Trans. Neural Networks*, vol. 4, pp. 740–747, September 1993.

- [51] G. Bebis and M. Georgiopoulos, "Feed-forward Neural Networks: Why Network Size Is So Important," *IEEE Potentials*, vol. , pp. 27-31, October/November 1994.
- [52] I. S. Duff, "A Survey of Sparse Matrix Research," *Proc. IEEE*, vol. 65, pp. 500-535, April 1977.
- [53] J. Vlach and K. Singhal, *Computer Methods for Circuit Analysis and Design*, Van Nostrand Reinhold Company Inc., New York, 1990.
- [54] K. S. Kundert, "Sparse Matrix Approach," In A. Ruehli, editor, *Circuit Analysis, Simulation and Design*, pp. 281-324, North-Holland, Amsterdam, The Netherlands, 1986.
- [55] *HSPICE User's Manual*, Meta-Software Inc., Campbell, CA 95008, 1992.
- [56] *Harwell Subroutine Library*, AEA Industrial Technology, Oxfordshire, England, 1990.
- [57] J. J. More and S. J. Wright, *Optimization Software Guide*, SIAM Publisher, Philadelphia, Pennsylvania, 1993.
- [58] P. E. Plassman, "Sparse Jacobian Estimation and Factorization on a Multiprocessor," In T. F. Coleman and Y. Li, editors, *Large-scale Numerical Optimization*, pp. 152-179. SIAM Publisher, Philadelphia, Pennsylvania, 1989.
- [59] J. W. Bandler and Q. J. Zhang, "An Automatic Decomposition Approach to Optimization of Large Microwave Systems," *IEEE Trans. Microwave Theory and Techniques*, vol. MTT-35(12), pp. 1231-1239, December 1987.

- [60] J. W. Bandler and Q. J. Zhang, "An Automatic Decomposition Technique for Device Modelling and Large Circuit Design," In *IEEE Int. Microwave Symp. Digest*, pp. 709-712, Las Vegas, Nevada, 1987.
- [61] Y. J. Wei, Q. J. Zhang, and M. S. Nakhla, "Multilevel Optimization of High Speed VLSI Interconnect Networks by Decomposition," *IEEE Trans. Microwave Theory and Techniques*, vol. 42(9), pp. 1638-1650, September 1994.
- [62] J. A. Freeman and D. M. Skapura, *Neural Networks: Algorithms, Applications, and Programming Techniques*, Addison Wesley, Reading, Massachusetts, 1992.
- [63] F. Wang and Q. J. Zhang, "An Improved K-Means Clustering Algorithm and Application to Combined Multi-Codebook/MLP Neural Network Speech Recognition," In *Proc. Canadian Conf. Electrical and Computer Engineering*, Montreal, Canada, September 1995.
- [64] R. P. Lippmann and B. Gold, "Neural-net Classifiers Useful for Speech Recognition," In *Proc. IEEE First Int. Conf. Neural Networks*, volume IV, pp. 417-425, San Diego, California, 1987.
- [65] N. Morgan and H. Bourlard, "Continuous Speech Recognition: An Introduction to the Hybrid HMM/connectionist Approach," *IEEE Signal Processing Magazine*, vol. 12(3), pp. 25-42, 1995.
- [66] D. P. Morgan and C. L. Scofield, *Neural Networks and Speech Processing*, Kluwer Academic Publishers, Boston, Massachusetts, 1994.

- [67] B. H. Juang(Chair), "Session: Applications of Neural Networks to Speech Processing," In *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing*, volume V, pp. 3307–3346, Detroit, Michigan, 1995.
- [68] L. Rabiner and B. H. Juang, *Fundamentals of Speech Recognition*, Prentice Hall, Englewood Cliffs, New Jersey, 1993.
- [69] H. A. Bourlard and N. Morgan, *Connectionist Speech Recognition: A Hybrid Approach*, Kluwer Academic Publishers, Boston, Massachusetts, 1994.
- [70] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang, "Phoneme Recognition Using Time-delay Neural Networks," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, pp. 328–339, 1989.
- [71] H. Hild and A. Waibel, "Multi-speaker / speaker-independent Architectures for the Multi-state Time Delay Neural Network," In *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing*, volume II, pp. 255–258, 1993.
- [72] P. Haffner, M. Franzini, and A. Waibel, "Integrating Time Alignment and Neural Networks for High Performance Continuous Speech Recognition," In *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing*, pp. 105–108, 1991.
- [73] Y. Bengio, R. De-Mori, G. Flammia, and R. Kompe, "Global Optimization of A Neural Network-hidden Markov Model Hybrid," *IEEE Trans. Neural Networks*, vol. 3, pp. 252–259, 1992.
- [74] L. T. Niles and H. F. Silverman, "Combining Hidden Markov Models and Neural Network Classifiers," In *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing*, pp. 417–420, Albuquerque, New Mexico, 1990.

- [75] Z. Huang and A. Kuh, "A Combined Self-Organizing Feature Map and Multilayer Perceptron for Isolated Word Recognition," *IEEE Trans. Signal Processing*, vol. 40, pp. 2651-2657, November 1992.
- [76] G. Kuhn and R. L. Watrous, "Connected Recognition with A Recurrent Network," *Speech Communication*, vol. 9, pp. 41-48, 1990.
- [77] S. J. Lee, K. C. Kim, H. Yoon, and J. W. Cho, "Application of Fully Recurrent Neural Networks for Speech Recognition," In *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing*, pp. 77-80, 1991.
- [78] Q. J. Zhang, F. Wang, and M. S. Nakhla, "A High-order Temporal Neural Network For Word Recognition," In *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing*, volume V, pp. 3343-3346, Detroit, Michigan, May 1995.
- [79] K. F. Lee, *Automatic Speech Recognition*, Kluwer Academic Publishers, Boston, Massachusetts, 1989.
- [80] S. Sandhu and O. Ghitza, "A Comparative Study of MEL Cepstra and EIH for Phone Classification under Adverse Conditions," In *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing*, volume I, pp. 409-412, Detroit, Michigan, 1995.
- [81] R. Gray, "Vector Quantization," *IEEE ASSP Magazine*, vol. 1, pp. 4-29, April 1984.
- [82] S. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Information Theory*, vol. IT-28(2), pp. 129-137, March 1982.

- [83] Y. Linde, A. Buzo, and R. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Communication*, vol. COM-28(4), pp. 84-95, January 1980.
- [84] C. Chinrungreung and C. Séquin, "Optimal Adaptive K-Means Algorithm With Dynamic Adjustment of Learning Rate," *IEEE Trans. Neural Networks*, vol. 6, pp. 157-169, January 1995.
- [85] S. A. Yarost, "A Circuit Extraction System and Graphical Display for VLSI Design,". Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB, Ohio, 1989.
- [86] M. A. Dukes, "A Multiple-Valued Logic Systems for Circuit Extraction to VHDL 1076-1987,". Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB, Ohio, 1988.
- [87] J. F. Wakerly, *Digital Design: Principles and Practices*, Prentice Hall, Englewood Cliffs, New Jersey, 2nd edition, 1994.
- [88] A. S. Sedra and K. C. Smith, *Micro-Electronics Circuits*, Sanders College Publishing, Fort Worth, Texas, 3rd edition, 1991.
- [89] F. H. Mitchell(Jr.) and F. H. Mitchell(Sr.), *Introduction to Electronics Design*, Prentice Hall, Englewood Cliffs, New Jersey, 2nd edition, 1992.

Appendix A

Instructions for Using the Sparse MLP Program

This program performs training and testing of multilayer perceptron neural network from user supplied training and testing data. The training technique is based on the sparse backpropagation training algorithm SEBP2, which can provide speed up in training over standard dense backpropagation, especially for neural networks which are not small.

I. USAGE OF THE PROGRAM

training:

```
mlp_training ConfigurationFile WeightsFile InputFile  
                OutputFile [ValidationInputFile ValidationOutputFile]
```

recognition:

```
recog ConfigurationFile WeightsFile InputFile OutputFile
```

II. INPUT AND OUTPUT OF THE PROGRAM "mlp_training"

Four (and optionally two more) files are needed to run "mlp_training":

1. ConfigurationFile

- 2. WeightsFile**
- 3. InputFile**
- 4. OutputFile**
- 5. ValidationInputFile (optional)**
- 6. ValidationOutputFile (optional)**

Two output files will be produced by running "mlp_training"

- 1. WeightsFile**
- 2. LogFile**

The output file "WeightsFile" is an updated version overwriting the input file of the same name.

II.1 Format of ConfigurationFile:

```

number_of_layers
number_of_neurons_in_layer_1
number_of_neurons_in_layer_2
...
number_of_neurons_in_layer_L
stop_criteria
number_of_epochs
cross_validation_flag
momentum_initial_value
learnrate_initial_value
adapt_learnrate_flag
learnrate_increase_ratio

```

`learnrate_decrease_ratio`
`sparse_threshold_for_input_layer`
`sparse_threshold_decrease_ratio`
`output_layer_sparse_threshold_lower_limit`
`hidden_layers_sparse_threshold_lower_limit`
`training_error_level_to_start_reducing_hidden_layers_sparse_threshold`
`sparse_refreshment_interval`

where `stop_criteria` is for absolute error threshold level, `number_of_epochs` is the interval to save the weights and perform cross validation if applicable. `Cross_validation_flag` is "1" for yes and "0" for no, `adapt_learnrate_flag` is "1" or "0" to enable or disable adaptation of η . `Sparse_refreshment_interval` is the initial value of the interval to perform sparse pattern refreshment.

II.2 Format of WeightsFile

Before running the program "mlp_training", this file contains the initial values of neural net weights. After running "mlp_training", this file contains the trained values of the neural net weights.

$$w_{10}^2 \ w_{11}^2 \ \dots \ w_{1n_1}^2$$

...

$$w_{n_20}^2 \ w_{n_21}^2 \ \dots \ w_{n_2n_1}^2$$

...

$$w_{n_L0}^L \ w_{n_L1}^L \ \dots \ w_{n_Ln_L-1}^L$$

II.3 Format of InputFile

```

string_1 x1 x2 ...xn
string_2 x1 x2 ...xn
...
string_P x1 x2 ...xn

```

where P is number of training patterns and n is number of inputs to the MLP. Each row of the file represents a training pattern. x_1 to x_n are the values of MLP inputs for a pattern. Each training pattern starts with a string which identifies the pattern.

II.4 Format of OutputFile

```

y1 y2 ...ym
y1 y2 ...ym
...
y1 y2 ...ym

```

where m is number of outputs of MLP. Each row represents a MLP output pattern. The number of output patterns should be the same as the number of training patterns, i.e., P .

II.5 Format of ValidationInputFile

Same as that for the InputFile, except that the data are validation data.

II.6 Format of ValidationOutputputFile

Same as that of OutputFile, except that they represent output patterns corresponding to validation data.

II.7 Contents of LogFile

epochs have passed!

Total_error is :

learnrate:

Update_Sparse:

Threshold for output:

....

epochs have passed!

...

III. INPUT AND OUTPUT OF THE PROGRAM "recog"

Four (optionally one more) files are needed to run "recog":

1. ConfigurationFile
2. WeightsFile
3. InputFile
4. OutputFile
5. NameFile (optional)

One output file will be produced by running "recog"

1. LogFile

III.1 Format of ConfigurationFile:

Same as that for program "mlp_training".

III.2 Format of WeightsFile

This file should be the updated version of WeightsFile after running "mlp_training".

III.3 Format of InputFile

Same as that for program "mlp_training", except that the data are testing data.

III.4 Format of OutputFile

Same as that for program "mlp_training", except that the data are testing data.

III.5 Format of NameFile

This file gives the meaning of each output neurons of MLP for pattern recognition application.

n_o

string_1

string_2

....

string_ n_o

where n_o means the number of neurons in the output layer of MLP.

III.6 Contents of LogFile

This file gives the recognition performance of the MLP by showing the list of recognition decisions for all the testing data and the statistical information such as how many patterns correctly recognized, wrongly recognized and the recognition

accuracy in percentage values.

END

0 5 - 0 6 - 9 6

FIN