Unsupervised Learning in Analog Networks

by

Dean K. McNeill

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

Master of Science

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

ISBN 0-315-85959-8

Name _____

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

ELECTRICAL ENGINEERING
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
- General 0679
- Ancient 0289
- Linguistics 0290
- Modern 0291

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

PHILOSOPHY, RELIGION AND THEOLOGY
Philosophy 0422
Religion
- General 0318
- Biblical Studies 0321
- Clergy 0319
- History of 0320
- Philosophy of 0322

Theology 0469

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

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

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

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

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

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

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

THE SCIENCES AND ENGINEERING

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

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

Biophysics
- General 0786
- Medical 0760

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

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

Home Economics 0386

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

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

Statistics 0463

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

Geotechnology 0428
Operations Research 0796
Plastics Technology 0795
Textile Technology 0994

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

Nom _____

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

SUJET

CODE DE SUJET

U·M·I

Catégories par sujets

HUMANITÉS ET SCIENCES SOCIALES

COMMUNICATIONS ET LES ARTS

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

ÉDUCATION

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

LANGUE, LITTÉRATURE ET LINGUISTIQUE

Langues
- Généralités 0679
- Anciennes 0289
- Linguistique 0290
- Modernes 0291

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

PHILOSOPHIE, RELIGION ET THÉOLOGIE

Philosophie 0422
Religion
- Généralités 0318
- Clergé 0319
- Études bibliques 0321
- Histoire des religions 0320
- Philosophie de la religion 0322

Théologie 0469

SCIENCES SOCIALES

Anthropologie
- Archéologie 0324
- Culturelle 0326
- Physique 0327

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

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

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

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

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

Transports 0709
Travail social 0452

SCIENCES ET INGÉNIERIE

SCIENCES BIOLOGIQUES

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

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

Biophysique
- Généralités 0786
- Medicale 0760

SCIENCES DE LA TERRE

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

SCIENCES DE LA SANTÉ ET DE L'ENVIRONNEMENT

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

SCIENCES PHYSIQUES

Sciences Pures

Chimie
- Généralités 0485
- Biochimie 487
- Chimie agricole 0749
- Chimie analytique 0486
- Chimie minérale 0488
- Chimie nucléaire 0738
- Chimie organique 0490
- Chimie pharmaceutique 0491
- Physique 0494
- PolymÇres 0495
- Radiation 0754

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

Statistiques 0463

Sciences Appliqués Et Technologie

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

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

PSYCHOLOGIE

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

UNSUPERVISED LEARNING IN ANALOG NETWORKS

BY

DEAN K. MCNEILL

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

MASTER OF SCIENCE

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

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

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

ABSTRACT

Artificial neural networks have shown their usefulness in the solution of a number of complex problems such as pattern recognition and associative memories. Typically these networks are simulated on serial computers or using expensive vector processors. This thesis examines several issues important to the implementation of unsupervised learning algorithms in compact dedicated analog structures. Two unsupervised learning algorithms known as Coherence Based Unsupervised Learning (CBUL) and Competitive Learning are discussed in detail.

A fully custom analog implementation of CBUL is presented and test circuitry implemented in 3μm CMOS is described. These circuits make extensive use of a CMOS version of the Gilbert multiplier to perform the majority of the neural computations. A complete implementation of the coherence based network, with capacitive weight storage, would consist of approximately 20 neurons and 400 synapses when fabricated on a 1cm silicon die in a typical 3μm technology.

Additionally, the effect of inherent device fabrication variations are examined in terms of their effects on the construction of analog circuits for competitive learning. Several simulations are conducted involving a number of modelled hardware effects, including multiplier gain variations, errors due to noisy multipliers, multiplier zero-crossing offsets, and the effects of noisy input signals. These results demonstrate that competitive learning is tolerant of most expected fabrication variations with the exception of multiplier zero-crossing offset errors.

ACKNOWLEDGEMENTS

I would like express my thanks to my advisor Professor Howard Card, whose continual enthusiasm, wealth of inspirations, and direction were invaluable in the completion of this thesis. I would also like to thank my friends and colleagues who have been a valuable source of information and assistance throughout my graduate program. Most notably, Bob Pelletier, Dave Blight, Chris Schneider, Bob McLeod, Zaifu Zhang, Martin Meier, Brion Dolenko, Brendan Frey, and Brad Brown.

Financial support for this work was received from NSERC, and from Micronet, a Network of Centres of Excellence, and was made possible with equipment loans and fabrication facilities provided through the Canadian Microelectronics Corporation.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

Introduction

Computers have become an indispensable tool in all areas of scientific research, and in fact have infiltrated virtually every facet of our daily lives. However, for all the amazing accomplishments in computer architectures and algorithms, there is a class of problems which have shown themselves to be exceedingly difficult, if not impossible to solve using standard computational techniques. These include such diverse tasks as speaker recognition, speech generation, image identification, pattern classification, character recognition, handwriting analysis, and the like. The reason these tasks are so difficult results from the fact that there is no clearly definable algorithm which one can lay out in order to accomplish them. Yet, in contrast, it is generally quite easy to produce large numbers of examples representing the problem at hand.

The extraordinary thing about this class of problems is that they are precisely the sorts of tasks which human beings have little or no difficulty doing, and in fact, we do them hundreds of times each day without a second thought. Some underlying biological factor has empowered us with the skills necessary to accomplish these feats; an aspect which appears to be lacking in traditional computer architectures. It would seem reasonable then to look at the source of our own knowledge and intellect, the human brain, for an insight into architectures which are more suitable for the solution of these varied problems. This is precisely what the field of neural networks, or connectionist research, attempts to do. It attempts to extract those features of biological learning which are essential for the solution of this suite of problems and to use the information in the design of algorithms and hardware architectures capable of performing these tasks.

The work presented in this thesis examines two specific learning algorithms and addresses some of the concerns associated with the eventual implementation of these algorithms in custom VLSI hardware. In the remainder of this chapter background material is presented which will serve to familiarize the reader with some of the general concepts associated with neural networks. This will help establish a common reference from which to base discussion in later chapters.

1.1 Basic Neural Network Theory

What is it about the human brain which makes it so much better at solving problems, such as those listed previously? A simple desktop computer is much faster and more efficient at performing arithmetic computations than the brain, so why is the brain so much more efficient in these other areas? An important reason is parallelism. The brain relies on a large network of simple, highly interconnected computing elements (neurons) for its computation, as opposed to a single complex processor in a serial computer. This parallel system, when coupled with a suitable learning algorithm, is the basis for solving the complex tasks we are interested in.

Connectionist research has borrowed from the structure of the brain, and has produced a simplified model which is felt represents the important features of biological computation. The result is an artificial neural network (ANN) computing architecture, the basic form of which is shown in Figure 1.1.

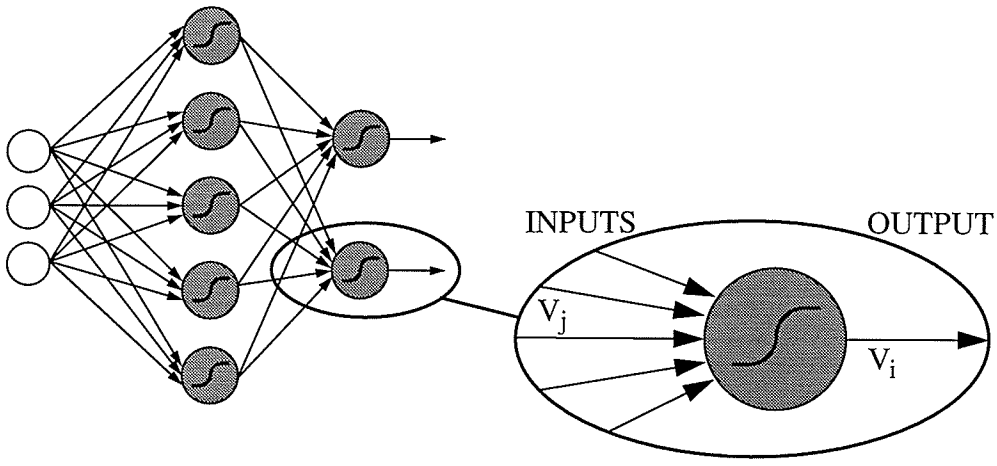

Figure 1.1: Basic neural network structure.

In keeping with their biological origin, each processing element in the ANN is called a *neuron*, and the connections between the neurons are known as *synapses*. A neuron may have many inputs, but produces only one output. Each synapse has associated with it a weighting which represents the contribution which that particular input has towards the neuron output. The neuron performs a simple weighted summation of its inputs and produces as its output, some nonlinear transformation of this input. This is represented mathematically in Equation 1.1.

$$V_i = f\left(\sum_j W_{ij} V_j\right) \tag{1.1}$$

Here V_i is the neuron output, V_j the inputs to the neuron, and W_{ij} the weighting associated with the connection from input j to neuron i. The individual inputs to a neuron may be supplied as either stimuli from an external source, or may be the outputs produced by other neurons in the network.

The nonlinear transformation $f(\bullet)$ of the weighted sum is commonly realized through the sigmoid function of Equation 1.2 in the majority of neural network algorithms. The shape of the sigmoid is shown in Figure 1.2 below.

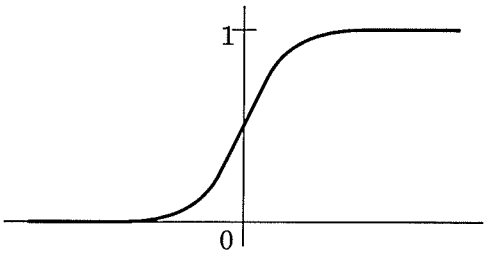

Figure 1.2: Sigmoidal nonlinearity.

$$f(x) = \frac{1}{1+e^{-x}} \tag{1.2}$$

Another common nonlinearity is the hyperbolic tangent function (Equation 1.3). It is used in place of the sigmoid function when it is desired to have the neuron output in the range [-1,+1] instead of [0,1].

$$f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{1.3}$$

In general, neurons are grouped into layers with each neuron in a layer receiving its inputs from the previous layer of neurons, and supplying its output to the following layer. Neurons which receive their inputs from an external source are said to be *input units* and are members of the *input layer.* Those that drive external signals are *output units*, and are members of the *output layer.* Neurons which are in neither the input layer nor the output layer are known as *hidden units* and are grouped into one or more *hidden layers.* The network in Figure 1.1 consists of three input units, five hidden units, and two output units, organized into three layers. The number of neurons in each layer of the network, and the number of layers depends on the application of interest and the type of network being used.

1.2 The Learning Algorithm

Neural networks differ from standard computers in a number of ways, but one of the most noticeable is the method of programming. Any traditional computer (both serial and parallel) performs its task by following a sequence of steps laid out by a human programmer in some form of programming language. Inputs are supplied by the user, and the program follows the prescribed set of steps in order to arrive at the final result. Neural networks, however, do not follow the standard programming model. Instead, they learn to solve the desired task by example, and not by explicit direction. All the information necessary to arrive at a correct solution for a given set of inputs, is stored in the values of the connection weights W_{ij}. It is through the modification of these weightings that the network is able to learn to perform the task assigned to it. The method by which the weights are modified is known as the *learning algorithm.*

In a newly constructed network the synaptic connection weights contain no useful information. Before the network can solve a desired problem for a specific input pattern, it must first be taught how to solve that problem in general. This is done by showing the network a set of sample inputs which are

representative of the task it must perform, and it is up to the learning algorithm to determine what connection weights are needed to accomplish it. Once trained, the network can then be shown new input patterns, from which it will produce outputs based on the these inputs and the learned connection weights. How effectively the network is able to combine the novel inputs and weights to produce a correct output is known as its ability to *generalize*. That is, if a network is presented with an input pattern which it has not been shown during the training phase, and it produces an output which would be considered correct within the bounds of the problem, then the network is said to show good generalization. If, conversely, the network is shown a new pattern, and it produces an output which would not be considered correct, it is said to show poor generalization. In practice, the measurement of generalization is not established on the basis of a single pattern classification, but is averaged over a large number of such inputs. This gives a more reasonable measure of the performance of the network.

Individual learning algorithms differ in the way in which information is encoded in the weights. Yet they all share an important common property. Knowledge encoded in a network is distributed among a number of weights of a number of neurons as a natural consequence of the learning process. This is an extremely valuable feature, since it will allow the network a level of tolerance to individual component failure. If a single synapse or neuron fails, the network should still function correctly, since only a small component of the distributed knowledge will be lost. The level of this robustness will depend on the size of the network, and the learning algorithm being used.

All of the numerous learning algorithms available can be grouped into two basic classes known as *supervised*, and *unsupervised* learning.

1.3 Supervised Learning

In supervised learning the network is trained by presenting a pattern at its input which is characteristic of the specific task. At the same time, information is provided indicating what the correct network output should be for that input case when the problem has been correctly learned. The input is propagated forward through the various layers until it reaches the outputs of the final layer. These generated outputs are then compared against the

desired output values, and the difference, known as the *error*, is then used as a basis for modifying the weights.This process is repeated hundreds or thousands of times, with each training iteration resulting in a small change in the weights. The objective of this training process is to minimize the amount of error between the desired and actual network outputs. If successful, the result will be a network which has learned a correct mapping from the input space to the desired output space. The process by which the network error is used in updating the weights is what makes the various forms of supervised learning unique. The best understood and most commonly used supervised learning algorithm is the *Backpropagation Algorithm*[13].

1.3.1 The Backpropagation Learning Algorithm

Backpropagation learning gets it name from the way in which the network error is used to update the weights. Under this scheme the resultant error between the actual and desired outputs is propagated in a backwards fashion from the output layer towards the input layer. The individual W_{ij} are modified during this process in proportion to their contribution to the output error. That is, the more influence a synaptic connection has on the eventual output, the more it is modified at each stage of learning. The network error is calculated via Equation 1.4, which is the mean-squared error of the outputs.

$$E = \frac{1}{2}\sum_{c}\sum_{i}(V_i - V_i^d)^2 \tag{1.4}$$

Here, V_i represents the actual neuron output, and V_i^d the desired or objective output. The summation index i ranges over the set of output units, and c over the set of training examples. Thus E is the accumulated error from all output units for the c training patterns presented.

This error is then used to adapt the weights by an amount calculated in Equation 1.5, which performs steepest (gradient) descent in E. If the neuron nonlinearity is the sigmoid function, then the value of $\partial E/\partial W_{ij}$ is given by Equation 1.6 for all weights in the output layer, and by Equation 1.7 for weights in the hidden layers [13][6]. Here V_i are the outputs of the neurons in the output layer, V_j are the output of the hidden units in the layer below the output layer, and V_k are the output of the next lower layer.

$$\Delta W_{ij} = -\varepsilon \frac{\partial E}{\partial W_{ij}} \tag{1.5}$$

$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial V_i} V_i (1 - V_i) V_j \tag{1.6}$$

$$\frac{\partial E}{\partial W_{jk}} = \left(\sum_i \frac{\partial E}{\partial V_i} V_i (1 - V_i) W_{ij} \right) V_j (1 - V_j) V_k \tag{1.7}$$

$$\frac{\partial E}{\partial V_i} = (V_i - V_i^d) \tag{1.8}$$

The ε term is known as the learning rate and is used to control the size of the weight updates. This factor is usually quite small, compared to other values in the network, and must be so in order for the network dynamics to settle smoothly. If ε is too large, the weight changes will become erratic, which will result in the network failing to learn the task.

If the neuron nonlinearity is implemented using the tanh(•) function instead of the sigmoid, then Equations 1.6 and 1.7 become [6]

$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial V_i} (1 - V_i^2) V_j \tag{1.9}$$

$$\frac{\partial E}{\partial W_{jk}} = \left(\sum_i \frac{\partial E}{\partial V_i} (1 - V_i^2) W_{ij} \right) (1 - V_j^2) V_k \tag{1.10}$$

Backpropagation does have its problems, however. The amount of computation that must be performed for each weight update increases with the number of weights and is approximately $O(N^3)$, where N is the number of weights in the network[8]. This is related to the fact that backpropagation networks are, most commonly, fully connected. This means that every output from any given layer is connected as an input to each neuron in the next layer. As a result, each successive neuron added to a layer will require the addition of more and more synaptic connections. In order to attempt to tackle realistic

problems, a large number of neurons and an even larger number of synapses are required. For example, one network[10] which attempts to identify handwritten digits utilizes 1000 neurons and 63660 synaptic connections. (Though this network has many fewer independent parameters as a result of the use of weight sharing.) This size of network takes an extremely long time to simulate, even on a RISC workstation, due to the shear volume of computations needed, even though the individual computations are very simple.

A great deal of effort has gone into attempts to modify the algorithm in order to speed up the learning process. However, the details relating to these enhancements are not important to the development of the work presented in this thesis. Those readers who are interested in a more in depth discussion of the backpropagation algorithm, the derivation of the learning equations, and its enhancements are directed to [13], [6], and [2].

1.4 Unsupervised Learning

Supervised learning depends on the presence of a labeled set of input data from which to train the network. However, for a variety of problems it is extremely difficult or impossible to produce a set of labelled training data that will fully characterize the input space the network may encounter. For these types of problems a learning algorithm is required that can adapt to new inputs as they appear, without the need for the data to be pre-labelled. Unsupervised learning algorithms fulfill this requirement. They differs from supervised learning in that there is no explicit indication to the network of what the correct outputs should be for any given input pattern. Instead, the network must decide what characteristics of the training set are relevant, and to modify the weights in order to extract those features. The method used to accomplish this is a function of the learning algorithm, which is based on some type of internal error measure. When a pattern is presented to the network, it is forward propagated to the outputs (through Equation 1.1), and the error measure is then applied to extract relevant statistics from these outputs. The connection weights are modified in order to minimize this objective measure, via the learning rule. The error measure may not be explicitly defined but may be implicit in the learning equation. At no time is the network given any external indication as to the desired output values.

1.4.1 Hebbian Learning

A very simple example of an unsupervised learning rule is known as Hebbian learning since it is based on observations by Dr. Donald Hebb[5]. One version of this says that inputs to a neuron, which are correlated with its output on a majority of the patterns presented, should have their weight increased so that these signals will agree more strongly in the future. Similarly, inputs which are not correlated should have their weights reduced in order to minimize their effect on the output. This idea is represented by Equation 1.11, where V_i in the neuron output, V_j is the input of interest, and ε is the learning rate.

$$\Delta W_{ij} = \varepsilon V_i V_j - b W_{ij} \tag{1.11}$$

The second term in this equation is a weight decay term and is used to prevent the weights from growing without bound. In the absence of a reinforcing input (which occurs when V_i and V_j are anti-correlated), this term will cause the weight to slowly decay towards zero, and the rate of this decay is controlled by the constant b[1].

It is important to note that the learning rule does not make reference to external knowledge of any kind. The incremental weight update is only dependent on the neuron output, the present weight, and the present input.

Though not explicitly determined in developing the weight update rule, the error function being optimized can be calculated. This equation is given in Equation 1.12, where V_j^{μ} and V_i^{μ} are the neuron input and output for the specific training pattern μ.

$$E\{W_{ij}\} = \frac{1}{2}\sum_{ij\mu}(bW_{ij}^2 - 2V_i^{\mu}V_j^{\mu}W_{ij}) \tag{1.12}$$

1. In the case where the inputs are bipolar, which occurs when the tanh(•) function is used, the decay term is not required for negative weight changes as in the case of unipolar inputs. However, it is still useful, since it allows the network to "forget" the effects of past weight updates by slowly reducing their effects over time. In dynamic environments this places more emphasis on recent inputs and reduces the influence of inputs which have not occurred recently.

1.5 Hardware Implementations of Neural Networks

As was stated earlier, the main focus of the work presented in this thesis is directed to the eventual implementation of neural network algorithms in hardware. Specifically, two different unsupervised learning algorithms are examined in relation to their suitability for construction in analog CMOS hardware. However, before discussing the details of these algorithms it is useful to explore the rationale for wishing to construct these hardware networks in the first place.

Much of the research performed in the field of neural networks is done using traditional serial computers which have been programmed to emulate the functions of the neural architecture being investigated. The problem with this approach is that, typically, training a network takes thousands of iterations through the training set, which translates into thousands of updates of hundreds/thousands of weights. In and of themselves, the computations are quite simple, but when taken as a whole, a powerful workstation can take days or even weeks to learn a particular task. In an effort to reduce this problem, researchers have attempted to speed up the learning process by simulating the networks on specialized vector processors. These parallel processors are generally quite expensive. Since the learning algorithms used in most networks are relatively simple, with localized computation, and the network structures themselves are highly parallel by definition, it would seem to make sense to construct specialized VLSI hardware, taking these factors into account. By constructing simple inexpensive processing elements which embody the desired learning algorithm, and using these specialized processors to construct a hardware network, the learning process can be greatly accelerated. As a consequence of this, the speed of processing after learning will also be increased, but this is often not of major concern, since only a single propagation is required to classify an input pattern.

So by implementing the architectures in hardware, we take advantage of the parallelism of the network structures. The most desirable situation would see the implementation of a complete network in a single VLSI chip, as this is both efficient in terms of ease of design and manufacture. In any hardware neural network it quickly becomes apparent that the limiting factor on the

number of neurons that can be fabricated in a single device is dependent on the size and quantity of the synaptic elements. For example, the network depicted in Figure 1.1 is made up of only 7 neurons, but has a total of 25 synapses. The more neurons a network contains, the more dominant the synapses will become in terms of the use of silicon area. Very quickly the amount of area required for the neuron circuitry is insignificant compared to the number of synapses feeding it. As a result, the synapse should be made as compact as possible, and it is primarily for this reason that an analog hardware implementation is preferred over digital structures. The amount of area required for equivalent computations using analog components is significantly smaller than what one would expect for a similar digital system.

Several groups, including [9], have designed single chip networks which will perform the forward propagation using analog components. However, these devices don't include circuitry for the learning algorithm, and as a result, the learning must be computed off-line by a serial computer. This allows for flexibility in the choice of learning algorithm, but does not help to reduce learning times since the learning must still be performed off-line by a serial computer. To truly take advantage of the parallel aspects of the network, both the forward propagation and the learning should be constructed in hardware. This will increase the silicon area required for each synapse, but will dramatically improve the learning speed, in addition to the forward propagation speed. This approach has been pursued by a number of researchers, including Chris Schneider of our laboratory[14].

For these reasons, it is felt that an analog architecture, with on-chip learning is the best arrangement for practical applications of neural network technology.

1.6 Summary

This chapter has provided a quick overview of the area of neural networks. It began by examining the biological motivation for the creation of these architectures, which was followed by a description of the artificial neural structures. The concept of a learning algorithm was presented, and was examined in relation to traditional programing techniques. Next, supervised networks were discussed, and specifically the algorithm known as Backpropagation.

This was followed by a similar examination of unsupervised learning, which is the main algorithmic area of interest to this thesis. Hebbian learning was then presented as an example of an unsupervised learning algorithm. Finally, the motivation relating to the need for dedicated hardware implementations of these algorithms was outlined, and specifically, the advantages afforded by the use of analog VLSI techniques.

In Chapter 2 the concepts of unsupervised learning will be extended as it relates to the hardware implementation of a specific unsupervised learning algorithm known as Coherence Based Unsupervised Learning. This algorithm will be presented from a theoretical perspective, and will then be examined in relation to a dedicated hardware implementation.

Chapter 3 examines a second unsupervised algorithm known as competitive learning. This method is presented, and various aspects relating to the potential hardware implementations of such a network are discussed.

In Chapter 4, conclusions will be drawn based on the simulations and measurements obtained in Chapters 2 and 3. In addition, proposals for future work in this area will be presented.

CHAPTER 2

Coherence Based Unsupervised Learning

In this chapter an unsupervised learning algorithm is examined known as Coherence Based Unsupervised Learning (CBUL) developed by Suzanna Becker and Geoffrey Hinton at the University of Toronto[7][1]. Initially the basic properties of this algorithm are presented, and this discussion is followed by a description of the work performed in attempting to implement this algorithm in fully custom analog CMOS hardware.

2.1 Coherence Based Unsupervised Learning

CBUL is an unsupervised learning method which attempts to train a neural network to extract higher-order spatially or temporally coherent information present in its input space. If the input is an image, for example, the network may be trained to recognize features such as the depth, reflectance, or surface orientation of an object. The training method used in CBUL is based on the assumption that the information which is present in neighbouring portions of the input space are relatively consistent, and that this fact can be exploited in order to extract the underlying higher-order properties of this input. The basic structure of a CBUL network is illustrated in Figure 2.1.

The network is made up of a number of small backpropagation style network modules which function independently of one another for the purposes of forward propagating the input patterns. Since the individual modules are small, the amount of computation performed in each one is also kept small. This reduces the learning time compared to networks such as standard backpropagation networks which are usually large fully connected, or very nearly

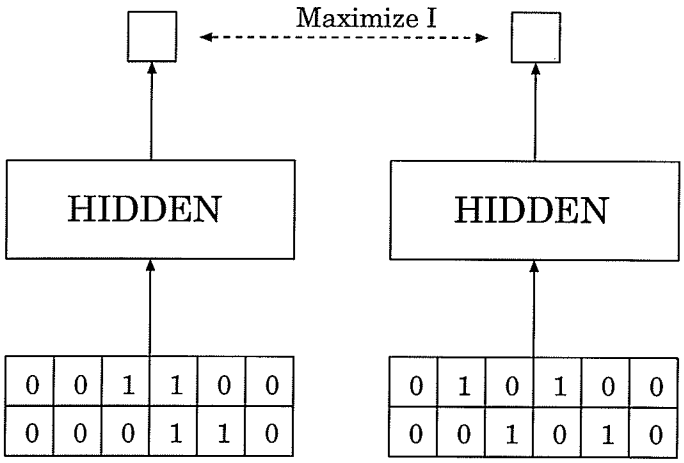

Figure 2.1: General structure of a CBUL network.

fully connected structures. Each of the layers in a module are fully connected to the following layer, but there are no connections between modules.

Coherence based networks differ from standard backpropagation networks in the method which is used to determine the output error. The technique employed here is based on the objective of assuring coherence between the outputs of adjacent modules, and does not make use of any external (supervisory) knowledge about the outputs.

But what kind of coherence method should be used? A very simple form of coherence would be to enforce the requirement that outputs of neighbouring modules be equal. There is a problem with this criteria, since the network will be able to fulfill this requirement perfectly by simply setting all weights to zero, and the resulting outputs will always be zero. This is obviously not a useful solution, so some other algorithm is needed to assure coherence. The answer is the use of a mutual information measure.

2.1.1 Mutual Information

Coherence based unsupervised learning attempts to solve the learning task by maximizing the mutual information present between adjacent output units. The mutual information measure used in this network is based on Equation 2.1, where $I(\bullet)$ is the mutual information, $H(\bullet)$ denotes the entropy of each input, and $H(a,b)$ is the entropy of the joint distribution.

$$I(a;b) = H(a) + H(b) - H(a,b) \tag{2.1}$$

In a network with continuous valued outputs, this can be written as[1]

$$I(a;b) = \log\frac{V(a)}{V(a-b)} + \log\frac{V(b)}{V(a-b)} \tag{2.2}$$

with $V(\bullet)$ denoting the statistical variance of the inputs. The simple interpretation of this equation is that in order to maximize the mutual information between the two inputs it is important that the individual outputs show a significant variation over time, while their mutual difference is kept as small as possible. This satisfies the coherence constraint while eliminating the trivial solution through the output variation requirement.

The derivative of Equation 2.2 is given by Equation 2.3, below.

$$\frac{\partial I}{\partial a^{\alpha}} = \frac{2}{N}\left[\frac{a^{\alpha} - \langle a\rangle}{V(a)} - 2\frac{(a^{\alpha} - b^{\alpha}) - \langle a-b\rangle}{V(a-b)}\right] \tag{2.3}$$

In this equation, a^{α} is the output of module a for the input pattern α, and b^{α} is the corresponding output for module b. $V(\bullet)$ denotes the variance of the specified output, $\langle\bullet\rangle$ represents a statistical average, and N corresponds to the number of patterns in the training set.

Two passes through the training set are required in order to update the weights. In the first pass, the average output and the variance of the outputs is accumulated. Then, on the second pass, the value of $\partial I/\partial a^{\alpha}$ can be calculated from the above equation, and the resulting value supplied as the error derivative for the weight updates in the small back-propagation modules.

2.2 A Sample Problem

An example of a simple CBUL network learning task is depicted in Figure 2.1. The objective of this network is extract the depth information present in a stereoscopic image. In a binocular system, such as the human visual system, depth information is determined by the amount of shift present between the images received by the left and right eyes. If either of the visual inputs is removed, a subject will no longer be able to determine depth in this manner

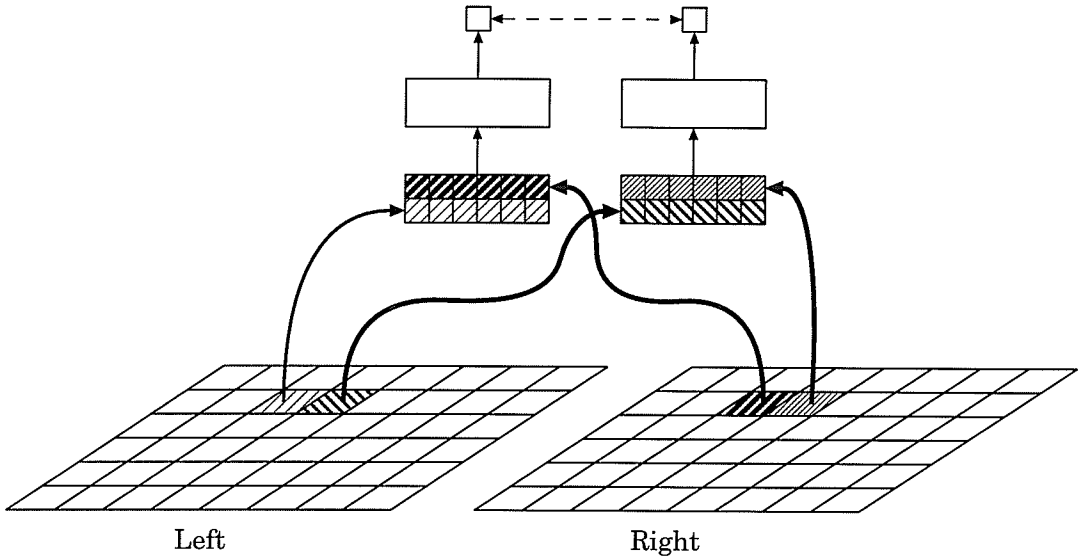

Figure 2.2: Stereoscopic input pattern selection.

and will have to rely on other factors in judging distances. The inputs to each module would be corresponding portions of a stereoscopic image, as represented by the diagram of Figure 2.2. The top row of the inputs is taken from the right image, and the bottom row from the left, or vice versa. In order to emulate this effect without the need to collect real stereoscopic images, the inputs to this sample network consist of a random binary bit stream with an artificially inserted bit shift. The shift inserted in the random stream corresponds to the depth in a real image. Since the input stream is random, except for the shift, the only consistent information that can be extracted to ensure coherence at the module outputs would be the bit shift, and hence the depth.

2.3 Hardware Implementation of the CBUL Algorithm

The objective of the work undertaken for this thesis was to translate the structure of the CBUL network into a comparable analog CMOS hardware implementation, thus taking advantage of the inherent parallelism of the architecture. This task can be broken down into two main areas; the circuitry to perform the backpropagation in the individual modules, and a circuit to calculate the mutual information measure.

The implementation of the backpropagation algorithm requires two basic

arithmetic computations. These are the multiplication of two parameters, and the addition of two parameters. If the parameters are represented in the form of currents, addition can be performed by summing of currents on a wire, employing the electrical properties known as Kirchoff's current law. However, in order to perform multiplication it is necessary to construct a multiplier using analog components. Such a multiplier was used previously by Chris Schneider of our laboratory in [14], and is known as the CMOS Gilbert multiplier, since it is an adaptation of a bipolar multiplier designed originally by B. Gilbert in [4].

2.3.1 The Gilbert Multiplier

The Gilbert multiplier is a transconductance multiplier producing an output current based on the product of two differential input voltages, as given by Equation 2.4. The term a controls the gain of the multiplier.

$$I_{OUT} = a\,(V_1 - V_2)\;(V_3 - V_4) \tag{2.4}$$

The measured response of our 3μm CMOS version of this multiplier for various ranges of inputs is shown in Figure 2.3. Each curve represents the characteristic for a particular value of the differential input (V_1-V_2), as the input (V_3-V_4) is varied. Over the input range +/-0.6 volts the multiplier is essentially linear, and does not show significant deviation until the +/-0.8 volt

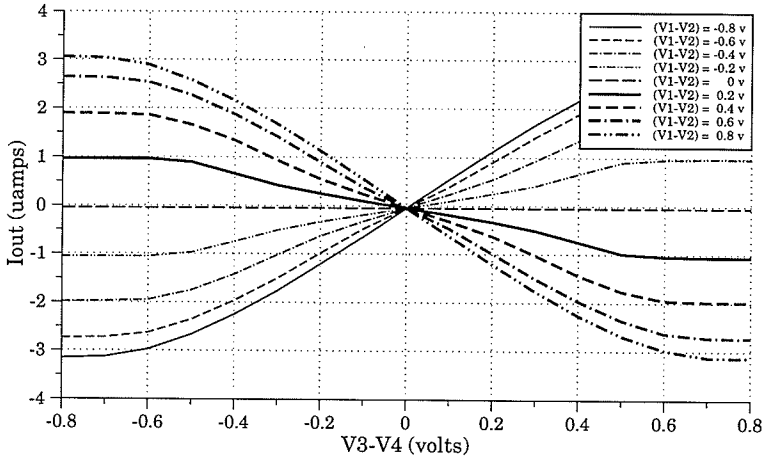

Figure 2.3: Gilbert multiplier characteristic.

level is reached. Outside of these ranges the output saturates. Minor variations in the multiplier characteristic are not considered significant, since the resulting product will ultimately be passed through the nonlinearity function in the neuron.

2.3.2 Circuit Implementation of Backpropagation

In addition to the Gilbert multiplier, a simple current/voltage converter is also required. The schematic representation of both of these components is given in Figure 2.4.

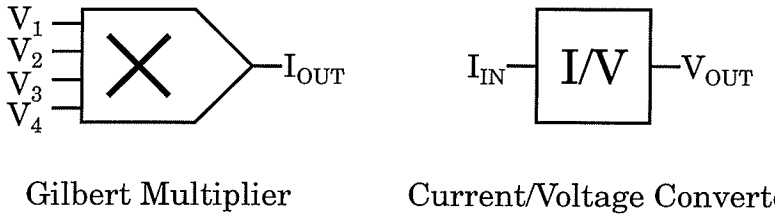

Figure 2.4: Hardware building blocks.

The hardware multiplier actually serves a secondary function in addition to multiplication. By exploiting its saturating characteristic it may also be used to implement the nonlinearity of the neuron output. This is important, as it reduces the number of components that must be designed, which helps to simplify the design structure. Figure 2.5 is a block diagram representation of the neural circuitry for a typical neuron/synapse in the hidden layer.

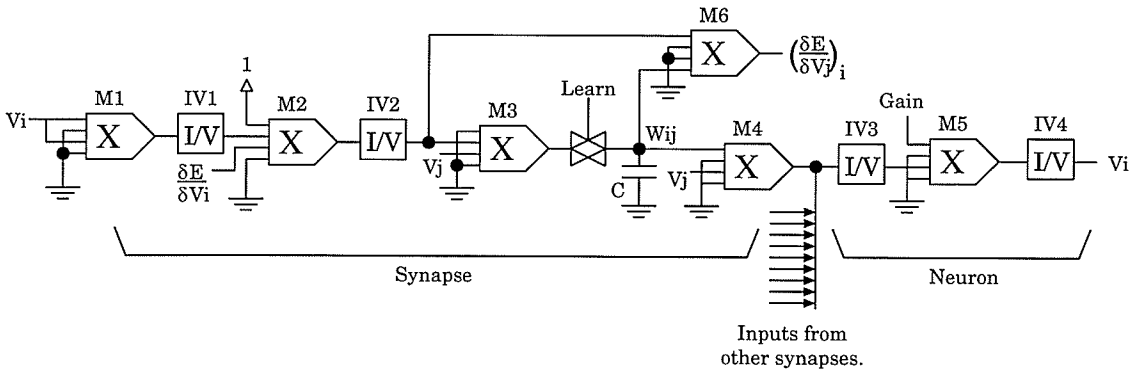

Figure 2.5: Block diagram of neuron backpropagation.

The functions of this circuit can be broken down into four sections. The first of these is the weight multiplier *M4* which computes the product of the synaptic input *Vj* and the weight W_{ij}. This weight value is stored as a quantity of charge on the capacitor *C*. Since this charge will leak away over time, thus changing the value of the stored weight, it is necessary to continually interleave training steps with classification steps in order to refresh its value. It would be greatly desirable to use some form of non-volatile storage instead of the capacitive storage, such as an EEPROM. However, this option was not supported in the available fabrication technology.

The output current from each synapse is summed with the output currents from the other synapses using the simple properties of Kirchoff's current law. This sum is then passed on to the neuron.

The neuron portion is made up of the two current/voltage converters *IV3* and *IV4* and the multiplier *M5*. *IV3* converts the summed synaptic input current into a corresponding voltage which then drives one input of the multiplier *M5*. The multiplier serves as the neuron nonlinearity under the control of the *GAIN* input. Increasing the *GAIN* input makes the neuron output more nonlinear, and also increases the range of the neuron output current. The resulting output current is converted to the neuron output voltage V_i by the current/voltage converter *IV4*.

The third section embodies the weight update Equation 1.9, as given in the backpropagation section of Chapter 1. This encompasses multipliers *M1–M3*, and convertors *IV1* and *IV2*. The input denoted "1", at the input to multiplier M2, represents the input voltage corresponding to a mathematical value of 1. The final output current of this section represents the weight update that is to be applied to the weight capacitor *C*. This update current is gated through the learning control circuitry which allows for the option of turning learning off and on. In practice the learning control is pulsed in order to update the weight, thus controlling the quantity of charge added to or removed from the capacitor. The duration of this pulse corresponds to the learning rate (ε) in Equation 1.5.

The final section is the weighted error calculation and is implemented by multiplier M6. This is the error which is passed back to the next lower hidden

layer below the current layer (if it exists) for use in updating the weights in that layer.

Two test implementations of this circuitry were designed using the Electric VLSI CAD software, and fabricated in 3μm and 1.2μm CMOS by Northern Telecom Electronics, through the auspices of the Canadian Microelectronics Corporation. A photomicrograph of the entire 3μm test layout appears in Figure 2.6, while Figure 2.7 shows a single synapse. Each synapse is 516μm by 526μm, while the entire test layout requires 7000μm by 2600μm. As can be seen this area is underutilized and is only required because of the relatively large number of test pads that are used in the design. The neuron shown in the figure does not include circuitry for the mutual information measure, but simply performs the nonlinearity operation on the weighted sum. The mutual information will be dealt with separately.

Simple tests were conducted to evaluate operation of the test implementation. The plot of Figure 2.8 shows the response of a single synapse to changes in its input. Here all inputs were held at a constant value except for the error input *ErrIn* which was switched between a positive and a negative input level. The resulting change in the value of the stored weight can be seen on the *SumOut* signal line. It is not possible to non-intrusively measure the weight value stored on the capacitor, so the effects of weight changes must be evaluated by observing their effects on the synaptic output. In the test, the synaptic output is initially at a strongly negative level, since the weight storage capaci-

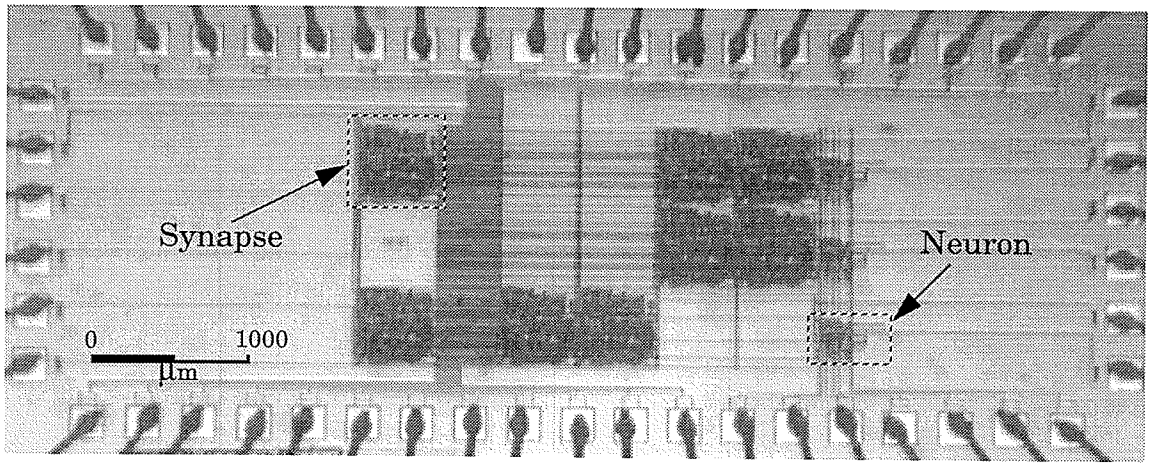

Figure 2.6: Photomicrograph of backpropagation test structure.

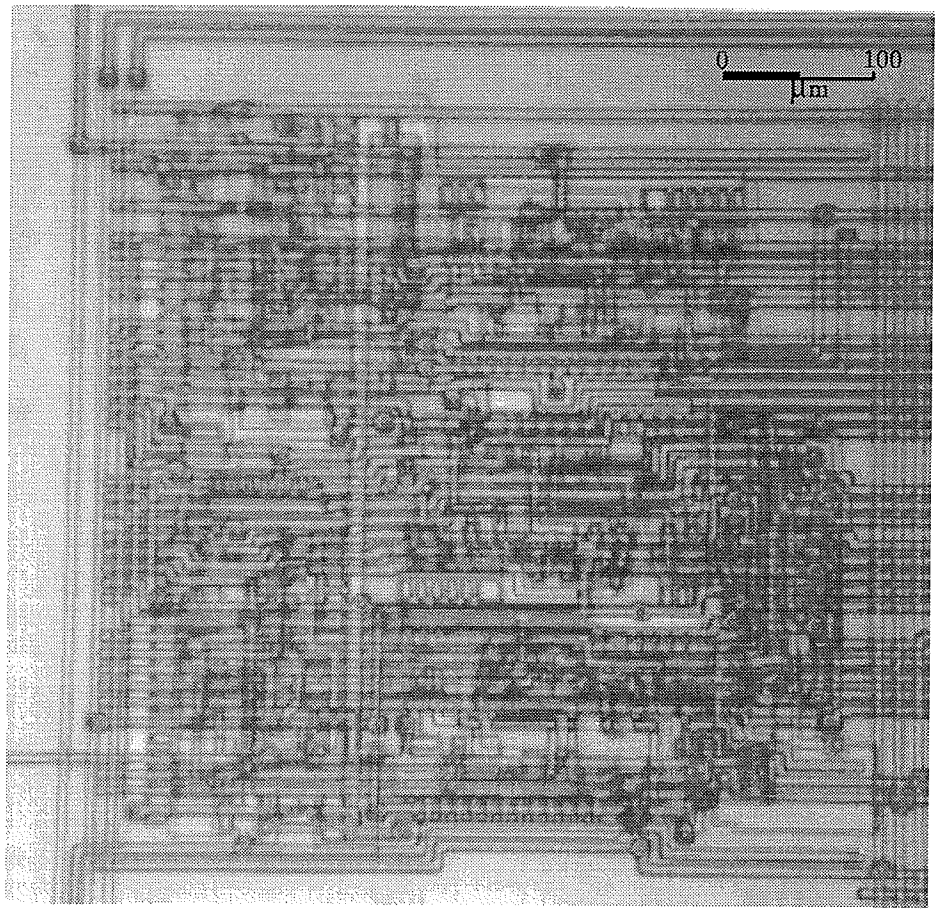

Figure 2.7: Photomicrograph of a synapse fabricated in 3μm CMOS.

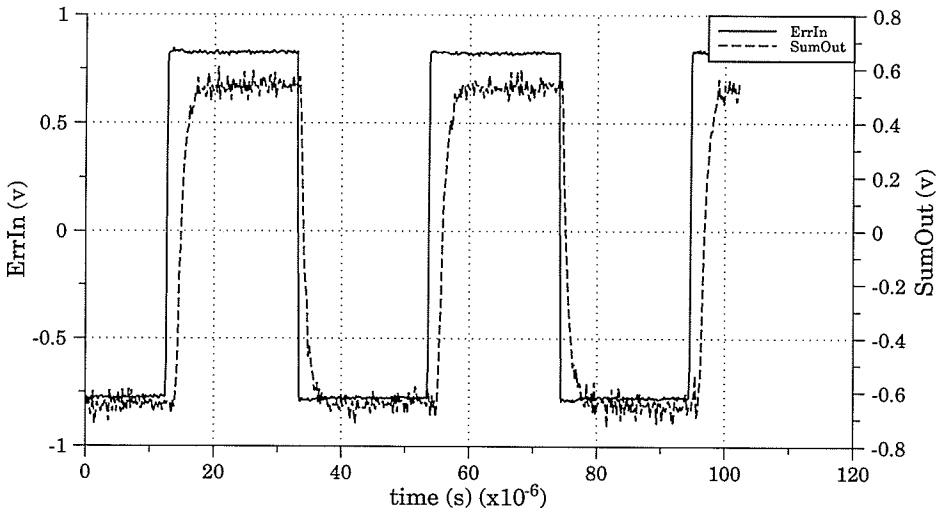

Figure 2.8: Measured synaptic characteristic.

tor is completely discharged. Switching the value of *ErrIn* to a strong positive value causes the weight update circuitry to produce a positive output current which charges the weight capacitor. This in turn causes the output of the synapse to become strongly positive, indicating that the weight capacitor changed from being fully discharged to fully charged. When the ErrIn value is once again switched low, the output becomes strongly negative as the weight capacitor once again discharges. This clearly demonstrates that the weight capacitor is being updated correctly with the changes on the input signals. In actual practice the update current to the capacitor would be gated in order to allow a small weight update for each training pattern, but for the purposes of these tests this was not done. The duration of the gating signal corresponds to the learning rate ε for the synapse since it controls the maximum amount of charge that can be added to the weight capacitor at each update in the training cycle.

It is clear that an implementation of the backpropagation style of learning will not be completely accurate in its implementation of the learning equations. The effects of using the Gilbert multiplier instead of a true arithmetic multiplier, as well as the component variations which will be encountered as a result of the fabrication process, will have an impact on the performance of the backpropagation learning. The effects of these factors as they relate to standard backpropagation learning have been examined by Brion Dolenko of our laboratory in [2]. His simulations showed that the synaptic circuits can be expected to perform correctly using the Gilbert multiplier and are able to tolerate anticipated variations in its gain. However it was also found that performance will suffer if significant multiplier zero crossing offsets are present in the system. So, based on the hardware tests and these simulations, the backpropagation style modules are expected to function correctly in the absence of multiplier offsets.

2.3.3 Circuit Implementation of Mutual Information

This section examines a proposed circuit for implementing the mutual information measure used by CBUL to generate the training error. Recall that this error is used by the backpropagation modules to update the synaptic weights. The error derivative has been given previously in Equation 2.3 and it

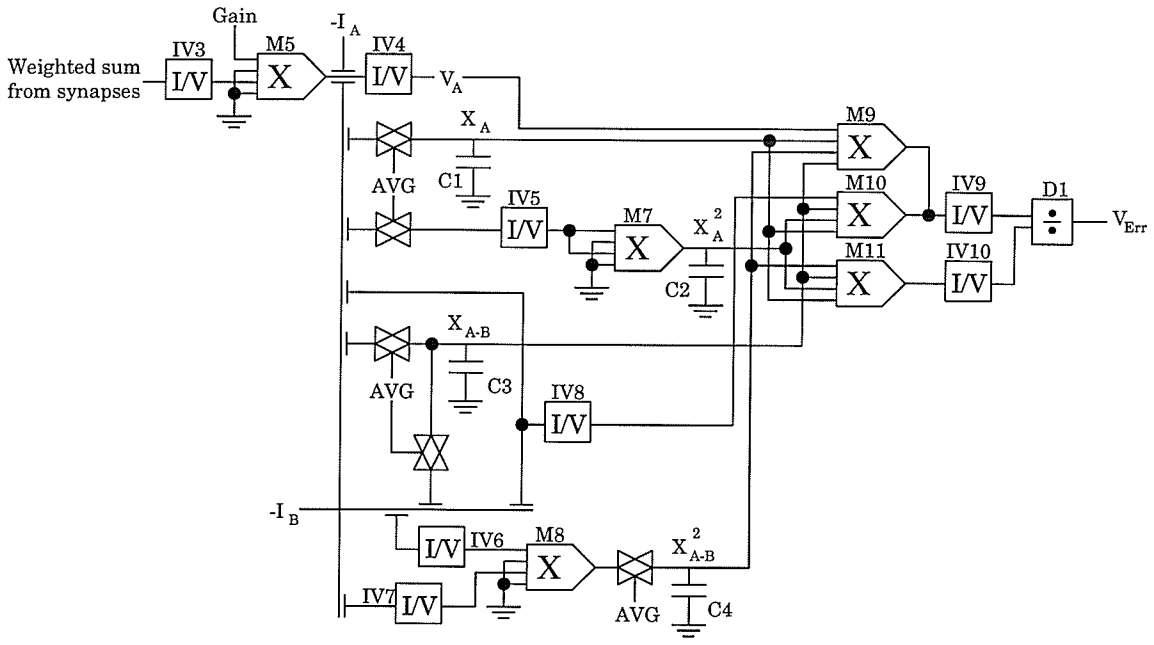

Figure 2.9: Proposed mutual information circuit.

is the objective of the following circuits to implement this equation in analog hardware. The diagram of Figure 2.9 shows the proposed circuit implementation. There are two major portions to this design; the circuitry to accumulate the statistics for the calculation, and the circuitry to perform the calculation itself. The statistical accumulation circuitry is comprised of the Gilbert multipliers *M7* and *M8*, and the current-voltage convertors *IV5–IV7*. The remaining circuit elements, *M9–M11*, *IV8–IV10*, and the voltage divider *D1* comprise the calculation portion. The elements labeled *M5*, *IV3*, and *IV4*, are the same three elements shown in the neuron output stage of Figure 2.5.

This circuit makes use of an additional symbol which has not been encountered thus far. This is the output stage of a current mirror and is shown in Figure 2.10 along with the equivalent analog circuit structure. The input stage of this mirror is already present within the Gilbert multiplier, and drives the multiplier output through a current mirror output stage. By using this circuit in conjunction with the control signals already present, it is possible to generate a second output current which is a copy of the first.

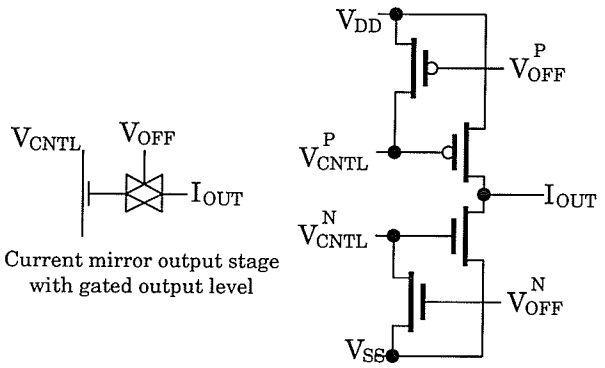

Figure 2.10: Current mirror output stage with control inputs.

The statistical accumulation portion of the mutual information circuit is required to collect a number of averages in order to generate the variances used in Equation 2.3. These are given by Equations 2.5–2.9.

$$X_A = \frac{1}{N}\sum_{\alpha=1}^{N} a^{\alpha} \tag{2.5}$$

$$X_A^2 = \frac{1}{N}\sum_{\alpha=1}^{N} (a^{\alpha})^2 \tag{2.6}$$

$$X_{A-B} = \frac{1}{N}\sum_{\alpha=1}^{N} (a^{\alpha} - b^{\alpha}) \tag{2.7}$$

$$X_{A-B}^2 = \frac{1}{N}\sum_{\alpha=1}^{N} (a^{\alpha} - b^{\alpha})^2 \tag{2.8}$$

$$\langle a \rangle = X_A \qquad V(a) = X_A^2 - X_A$$
$$\langle a-b \rangle = X_{A-B} \qquad V(a-b) = X_{A-B}^2 - X_{A-B} \tag{2.9}$$

The simplest of these averaging circuits computes the mean of the neuron output over the range of training cases. It is denoted X_A, and its value is stored on capacitor *C1*. The *AVG* signal is used to control the amount of charge

added to the capacitor, and as with the gating circuit used in the synapse, it is pulsed for a fixed duration for each training input. The pulse duration corresponds to the $^1/_N$ term of the average. The second sub-circuit stores the mean of the square of the neuron output X_A^2 on capacitor *C2*. The squaring operation is carried out by multiplier *M7* and the resulting output is transferred to the storage capacitor. Similarly the average difference between adjacent outputs $X_{A\text{-}B}$ and the squared difference $X_{A\text{-}B}^2$ are stored in the same manner on capacitors *C3* and *C4,* respectively.

All four of these stored averages are accumulated on the first pass through the training set of N patterns. The actual weight updates are then performed on the second pass where the stored averages are combined with the current output of the synapse to generate the final error value $\partial I/\partial V_i$, which is denoted V_{ERR} in the circuit. This is the error value which is passed as an input to the output layer backpropagation style synapses.

Though this circuit appears to be much more complex than the synaptic circuit of Figure 2.5, it is actually comparable in size, given the number and type of elements used. In order to evaluate the feasibility of this structure it would be necessary to generate a hardware layout for simulation and fabrication purposes. However, based on experience with the layout of the elements used, we can readily estimate that the resulting layout of the neuron would be on the order of 1.5–2 times the area occupied by one of the backpropagation synapses. However, there are far fewer neurons in a network than synapses, so a neuron of this size is not unreasonable. The complete neuron portion of the circuitry (all elements from Figure 2.9) would occupy an area of approximately 1000μm by 500μm in a final implementation. Using a 3μm fabrication technology this would allow for approximately 20 neurons and 400 synapses on a typical silicon die measuring 1cm on a side.

2.4 Summary

In this chapter an unsupervised learning algorithm based on coherence between adjacent spatial portions of the input data has been discussed in its relation to the solution of a simple depth extraction problem. As well, circuitry to perform the backpropagation of errors used in the hidden layers has been designed and fabricated using CMOS VLSI analog design techniques. In addi-

tion, a proposed circuit to calculate the mutual information error measure for controlling the backpropagation weight updates has been given. Both of these circuits make extensive use of the CMOS wide-range Gilbert multiplier, and capacitive weight storage.

CHAPTER 3

Competitive Learning

This chapter deals with some of the problems which are encountered when implementing networks in hardware, and specifically how these difficulties effect competitive learning algorithms[6]. Initially the theory of competitive learning will be explored, and this information will then be expanded upon as it relates to potential analog hardware implementations of this algorithm.

3.1 Competitive Learning

Competitive learning (CL) is an unsupervised learning procedure that attempts to motivate correct learning through the use of competition between output units. The basic structure of a simple competitive learning network is shown in Figure 3.1.

In addition to the layer to layer connections seen in other networks, CL networks also have connections from each output unit to all output units, including the source unit itself. The connection from a unit to itself is excitatory, meaning that it has a positive weighting and will help to reinforce the output of the neuron. The connections to all other output units are inhibitory and attempt to suppress the output of the other neurons. It is this mechanism which is the basis of the network competition. (Note: For simplicity, not all connections between output units have are shown in the figure.)

It is the objective of competitive learning to modify the connection weights in order to have each output, or group of outputs, represent some underlying class in the input data. As a result of this behavior, these networks are also commonly known as clustering networks, since they cluster inputs into a

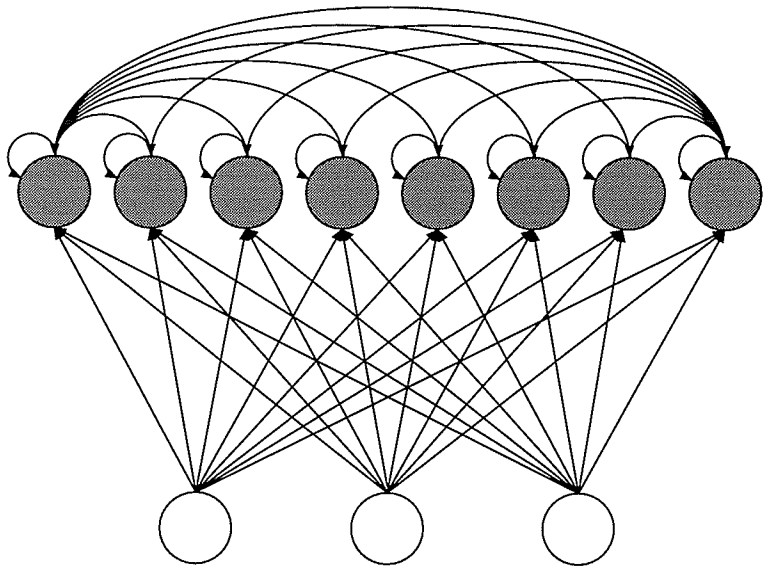

Figure 3.1: Basic competitive learning network.

number of categories. Ideally each input presented should excite a single output unit strongly, while all other outputs remain low. This is an indication that the network is confident as to the class membership of the input.

3.2 Hard and Soft Competitive Learning

There are two general classes of competitive learning, hard competitive learning (HCL) and soft competitive learning (SCL).

In hard competitive learning ideally only one neuron output should be active at any one time. An input pattern is applied to the network and the outputs are allowed to settle to stable values. The unit whose output is most active for this pattern is selected as the "winner" for this input, and this unit then updates its connection weights in order to reinforce the response. The weights connected to all other output units remain unchanged. As a result, the winning unit will be more likely to win for this input in the future. The forward propagation of input patterns to outputs is performed by Equation 1.1.

The weight update rule for HCL is given by Equation 3.1 for normalized inputs. Here W_{i^*j} denotes the weight connecting input unit j to the winning neuron i^*, V_j is the jth element of input pattern V^μ, and ε is again the learning

rate. Initially the weights are set to small random values before training.

$$\Delta W_{i^*j} = \varepsilon (V_j^{\mu} - W_{i^*j}) \tag{3.1}$$

There is a problem with this method of weight updating which manifests itself if the random initial weights place a unit outside the range of the input patterns. If this occurs, this particular unit will never be selected as the winning unit for any of the input patterns, and as a result, its weights will never be updated. These orphaned units will, therefore, never contribute to the solution derived by the network and are essentially wasted. An alternative learning method which combats this problem is known as soft competitive learning.

In soft competitive learning, all output units update their weights in relation to their output level. The unit which would be classified as the winner updates its weights the most, and all other units update their weights in proportion to their reduced activations. This ensures that all units get utilized in the eventual clustering solution by slowly moving orphaned units into the solution space. Eventually, these units will start to win for some of the input patterns, and the result is more efficient utilization of the outputs. The learning equation for SCL is given by Equation 3.2 below.

$$\Delta W_{ij} = \varepsilon V_i (V_j^{\mu} - W_{ij}) \tag{3.2}$$

The only difference between this equation and that of Equation 3.1 is the addition of the V_i term which is the output value of the unit being updated. This equation is similar to the weight update equation for Hebbian learning employing a weight decay term (Equation 1.11).

It should be noted that the update rule of Equation 3.2 was not derived from a previously determined error measure, as was the case for backpropagation in Chapter 1. However, a corresponding error measure does exist and is given by Equation 3.3[6].

$$E\{W_{ij}\} = \frac{1}{2}\sum_{ij\mu} V_i^{\mu} (V_j^{\mu} - W_{ij})^2 \tag{3.3}$$

Gradient decent based on this error function will result in the weight update function of Equation 3.2.

3.3 Hardware Issues of Competitive Learning

As was the case with Coherence Based Unsupervised Learning it is our intension to examine the competitive learning algorithm in relation to its eventual implementation in analog CMOS hardware. This and following sections will explore the feasibility of CMOS implementations in relation to expected process variations, based on experience with CMOS circuitry measurements such as those of Chapter 2. As was the case with CBUL the major component of the proposed hardware circuits will be our CMOS version of the Gilbert multiplier.

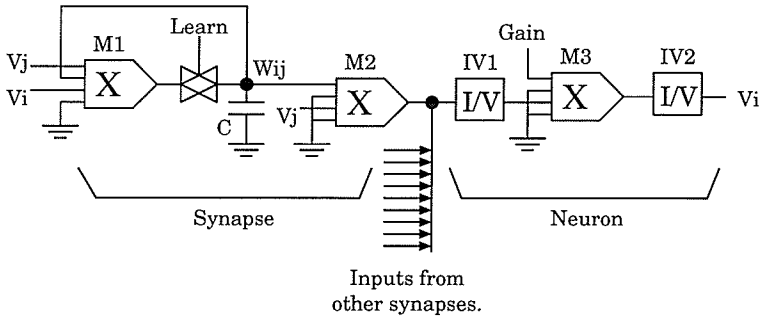

Figure 3.2: Proposed competitive learning circuit implementation.

A diagram of the proposed synapse/neuron circuitry appears in Figure 3.2 above, and is very similar to the synaptic circuitry used in CBUL in Chapter 2. The learning algorithm of Equation 3.2 is realized by a single multiplier *M1*, while the remainder of the circuitry remains unchanged.

3.4 Effects of Fabrication Variation on Learning

Though in theory it is possible to implement these unsupervised learning algorithms in hardware, the actual fabrication process will introduce variations in the characteristics of the fabricated components. As a result, the fabricated components will not operate precisely as predicted by theory, but will show a slight variation in their characteristics. The following sections will investigate the effects that these variations will have on the learning process. This will help to determine whether the competitive learning algorithm can

tolerate them, as well as showing how the variations affect parameters in the learning equation. The basis for the evaluations given are the direct result of a number of simulations of the CL algorithm using the Xerion neural network simulator[15], which was modified for the purposes of these hardware simulations. During simulation the characteristics of the Gilbert multiplier are modelled using a trigonometric hyperbolic tangent function. This function is commonly used to implement the nonlinearity in some neural network implementations as discussed earlier. However, it also compares closely with the measured characteristics of our CMOS hardware multipliers of Figure 2.3. Both the multiplier characteristics and tanh(•) function saturate at the extremities of the input range, as well as being essentially linear near the midpoint of this range. Thus it is felt that parameterized versions of tanh(•) functions serve as good models of the hardware multipliers.

In order to determine whether competitive learning is a good candidate for eventual fabrication in hardware it is important to explore the range of fabrication variations which this algorithm will tolerate. Once this is known it can be compared with the known parameters for the fabrication technology and a determination of its suitability can be well established.

In order for this evaluation to be carried out, it was necessary to select some task for the network to perform, and to be evaluated on. The problem selected involves the learning of a simple binary decoder. The basic structure of the decoder network is identical to that given previously in Figure 3.1. Each input pattern is a three bit binary number, and if correctly trained, the network should map each of these inputs to a single output unit. Thus, when an input is presented only one output should be activated. As a result the connection weights for each output neuron, when taken as a group, should represent a logical minterm. An example of actual weight values results for a correctly learned decoder problem are shown in Figure 3.3.

The connections to each of the eight output neurons is represented by the eight outer boxes. Each of the three smaller boxes within these represents the connection strength of one weight. The sign of the weight is indicated by the colour of the fill in the weight boxes, with positive values being white, and negatives being black. The magnitude of the weight is represented by the amount of the box which is shaded black or white. The group of three weights

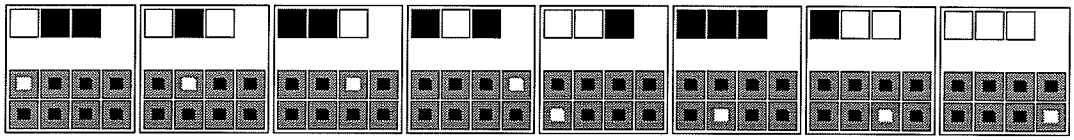

Figure 3.3: Xerion weight connection display.

at the top of each neuron box are the weights from the three input units to the particular output unit. The eight weights grouped below these are the connection weights from all other output units to the given unit. These eight weights are held constant during the training process, while the three input weights are varied by the learning algorithm. Note that the weights in Figure 3.3 correspond to soft competitive learning, and that when learning is complete the weights have taken on maximal positive and negative values.

3.4.1 Variations in Multiplier Characteristics

Device variations are an inevitable result of the VLSI fabrication process. These variations are not as critical for the production of digital circuitry because of the relatively high tolerances which are permitted before faulty operation is experienced. However, analog components are much more susceptible to process variations, and as a result these variations must be accounted for in the design process. The variations introduced during fabrication may result in a number of perversions of the ideal device characteristics. Though every effort was taken in the layout of the Gilbert multiplier circuitry to minimize the effects of these variations, they can not be completely eliminated and will result in variations of the multiplier characteristics. As a result a number of simulations of the competitive learning algorithm of Equation 3.2 were carried out in order to identify the effects of this variation.

The first of these potential effects is the replacement of the arithmetic multiply with the nonlinear hardware multiplier itself. Will the network be able to learn under these circumstances? Figure 3.4 shows the results of simulations using an ideal arithmetic multiplier, while Figure 3.5 shows the same simulation using the tanh multiplier. Twenty learning trials were attempted using a variety of learning rates (ε) in the range (0,0.1). The training inputs

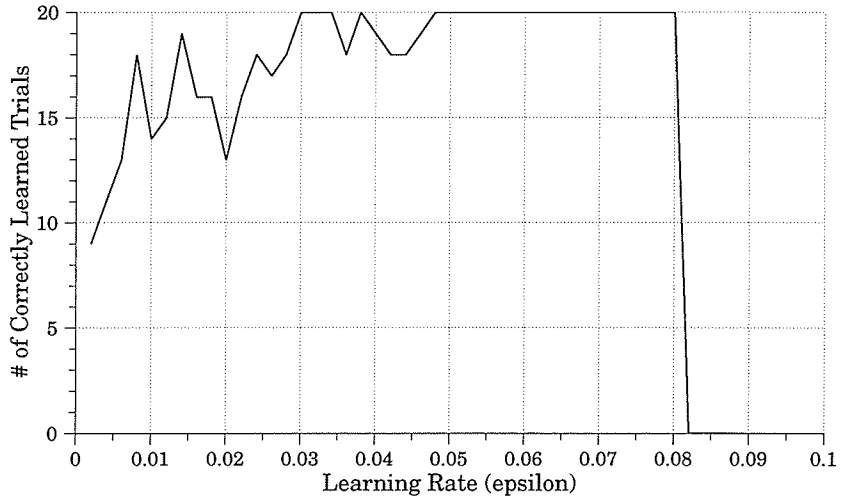

Figure 3.4: Learning using the ideal arithmetic multiplier.

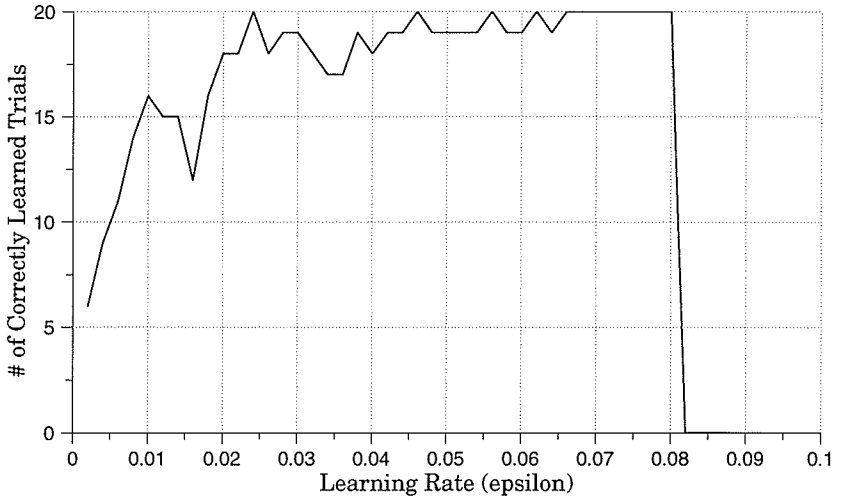

Figure 3.5: Learning using tanh multiplier model.

for the network consist of the complete set of eight possible binary input patterns, which fully characterize the range of possible input states. Since the training inputs and the testing inputs comprise the same set of patterns generalization is not an issue. The only factor that was changed between separate learning trials at any given learning rate was the random values of the connection weights. All other factors remained constant.

It is clear from these plots that the tanh network is able to reliably learn the task for a wide range of learning rates, and that the use of the non-ideal multiplier does not significantly affect the learning process. Best learning performance is found over the range ε=(0.05, 0.08). Learning rates higher than 0.08 result in larger weight modifications at each learning step, which cause the weights to become unstable. As a result, the network does not settle to a stable state, and will produce an unsatisfactory solution. Learning rates which are too small will result in the network becoming trapped in a non-optimal solution, from which it is unable to escape. Selecting a learning rate midway between these two extremes will eliminate both these problems.

In the cases where learning becomes unstable the final weights contain no useful information. However, in those cases which result in non-optimal solutions the final weight values do embody some usable information. In this case most output units have learned weights which represent a particular minterm. Those that do not have learned weights which represent a sum of minterms, instead of a single minterm. As well, a non-optimal solution can result in which two output learn the same minterm value. For the purposes of the simulations conducted, learning is only considered successful when all eight output units learn a unique minterm.

3.4.2 Effects of Multiplier Gain Variations on Learning

It has been shown that the arithmetic multiply can be safely replaced by a non-ideal Gilbert multiplier. However, it must also be determined how variations in this multiplier will effect the learning process. To evaluate the effects of variations, the soft competitive learning module of the Xerion simulator was further modified to allow for the introduction of statistical variations in the multiplier gain. Variations in gain are equivalent to changing the slope of the multiplier characteristic. The variations introduced follow a gaussian distribution and are applied to the characteristic of the multiplier in order to perturb the gain (or slope) of the multiplier characteristic. This multiplier is used in the computation of the weight update value in the learning equation. A normal multiplier characteristic has a slope of 1, and these random variations alter that slope by various amounts. Figure 3.6 gives the results of simulations using a variety of slope variations. Each plot in this figure represents the

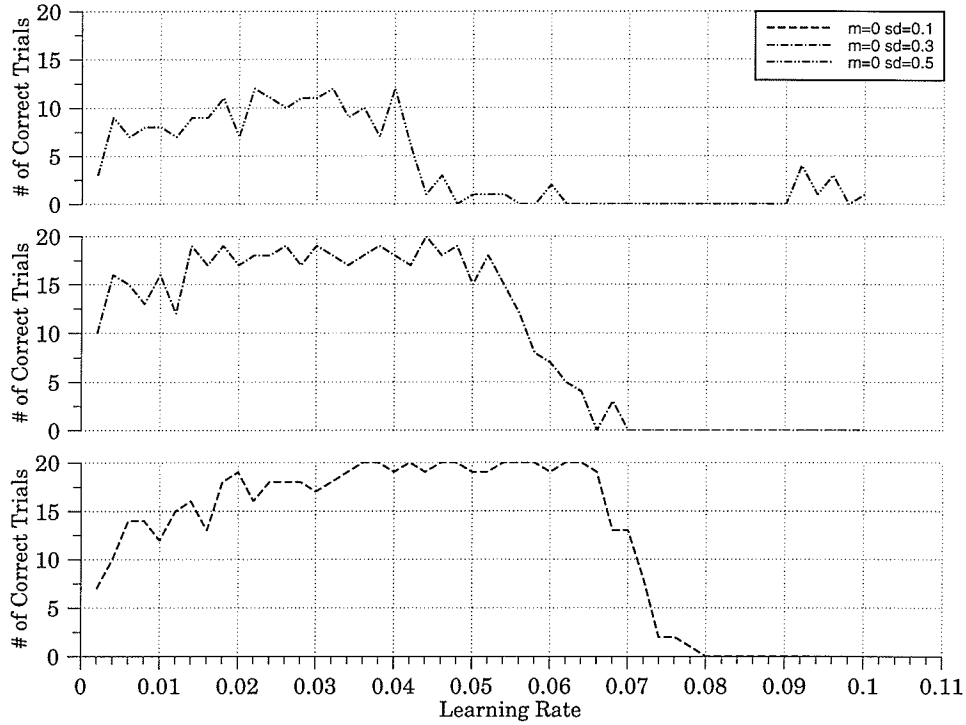

Figure 3.6: Non-ideal multiplier with uniform slope variations

number of correctly learned training runs from the twenty learning trials attempted at each value of ε. At the start of a training run each synaptic multiplier was assigned a modified slope selected from a gaussian distribution centred at 1. The standard deviation of the distribution was changed between plots, but remains constant in any given plot. The bottom plot shows the results with a small standard deviation of 0.1, while each successive plot shows the same simulation with a larger and larger variation.

Initially, the network is able to tolerate the gain variations, with only a small reduction in the acceptable range of learning rates when compared to the unaltered tanh multiplier of Figure 3.5. Over the range ε=[0.02,0.06] the network is able to learn correctly in 95% of the learning trials performed. As the range of variations increases the range of acceptable learning rates shrinks. For a standard deviation of 0.3 the network in still able to learn correctly 90% of the time, but the range of ε has been reduced to [0.02,0.05]. As degradation of the multiplier continues the network is soon unable to adequately compensate for this behaviour. At a standard deviation of 0.5 the net-

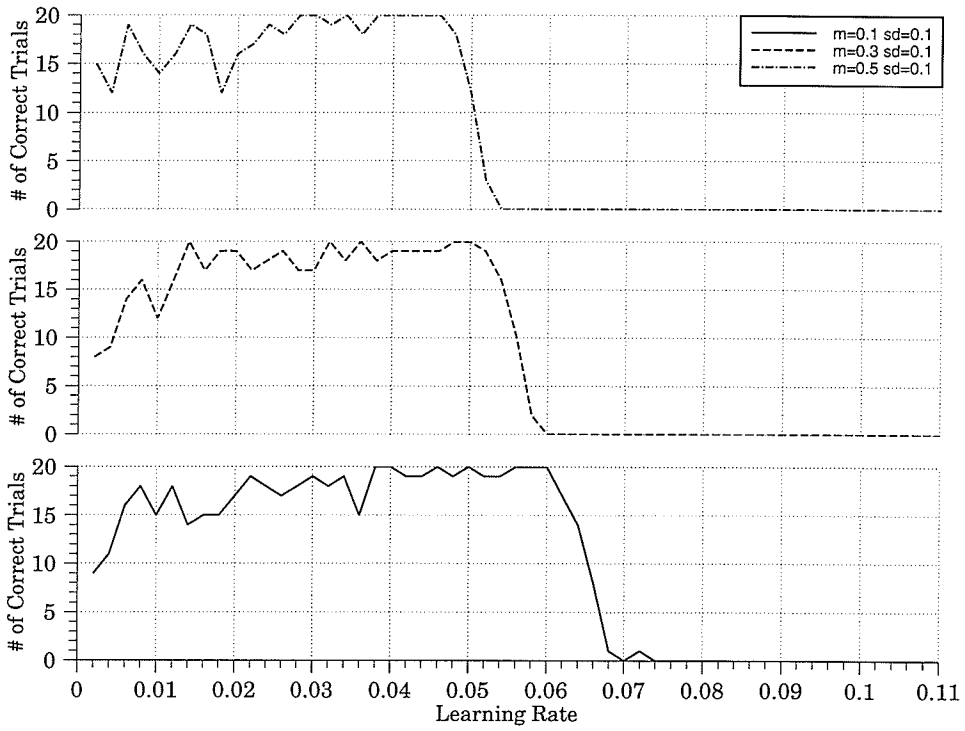

Figure 3.7: Non-ideal multiplier with positively biased slope variations

work is only able to achieve correct learning in 51% of the trials, over the range ε=[0.02,0.04].

Additional simulations were also carried out to determine the effects of non-uniform slope variations. In this case, multipliers are considered where the mean of the gaussian perturbations is not 1, but is shifted in either a positive or negative direction. Figures 3.7 and 3.8 show the results of these simulations using a standard deviation of 0.1 (corresponding to the bottom plot of Figure 3.6), and various mean offsets.

For the positively biased variations in Figure 3.7 the network is able to compensate quite well for the change in slope. The only significant change between the separate plots is the reduction of the range of valid learning rates. This reduction is in response to the corresponding increase in the slope of the multiplier characteristic which is producing larger products. Over this reduced range the network is still able to learn the task correctly in 94% of the trials shown.

A similar result is observed for simulations involving the negatively biased variations of Figure 3.8. Here the range of viable learning rates has expanded

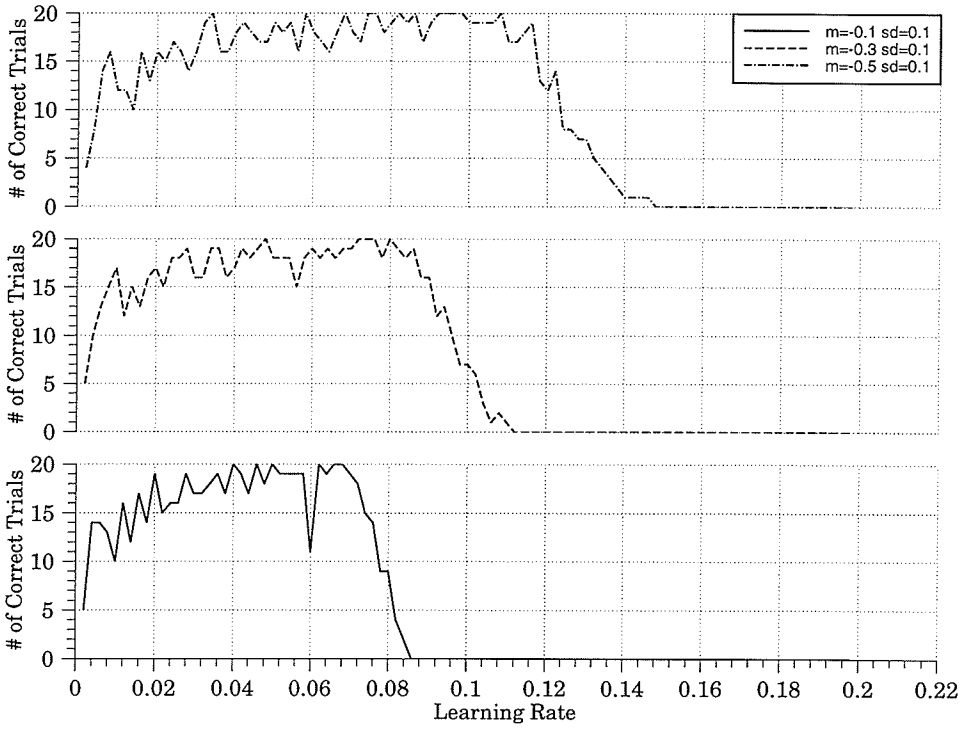

Figure 3.8: Non-ideal multiplier with negatively biased slope variations

in relation to the variation in slope, which is to be expected with the corresponding reduction in the multiplier characteristic. As a result, the network is able to find a correct solution in 91% of the above trials, within the active range of the results.

These simulations demonstrate that the network learning algorithm is capable of compensating for multiplier slope variations, as long as these variations are not too severe. Experimental hardware variations for actual fabricated devices have resulted in approximately 10% variation in the multiplier gain[14]. These variations are well within the acceptable boundaries as derived from the simulations. So slope variations are not expected to have a significant effect on network performance in an actual hardware implementation.

3.4.3 Effects of Circuit Noise on Learning.

Another area which was explored deals with the effect noisy circuitry has on the learning process. In order to determine the impact of this phenomenon,

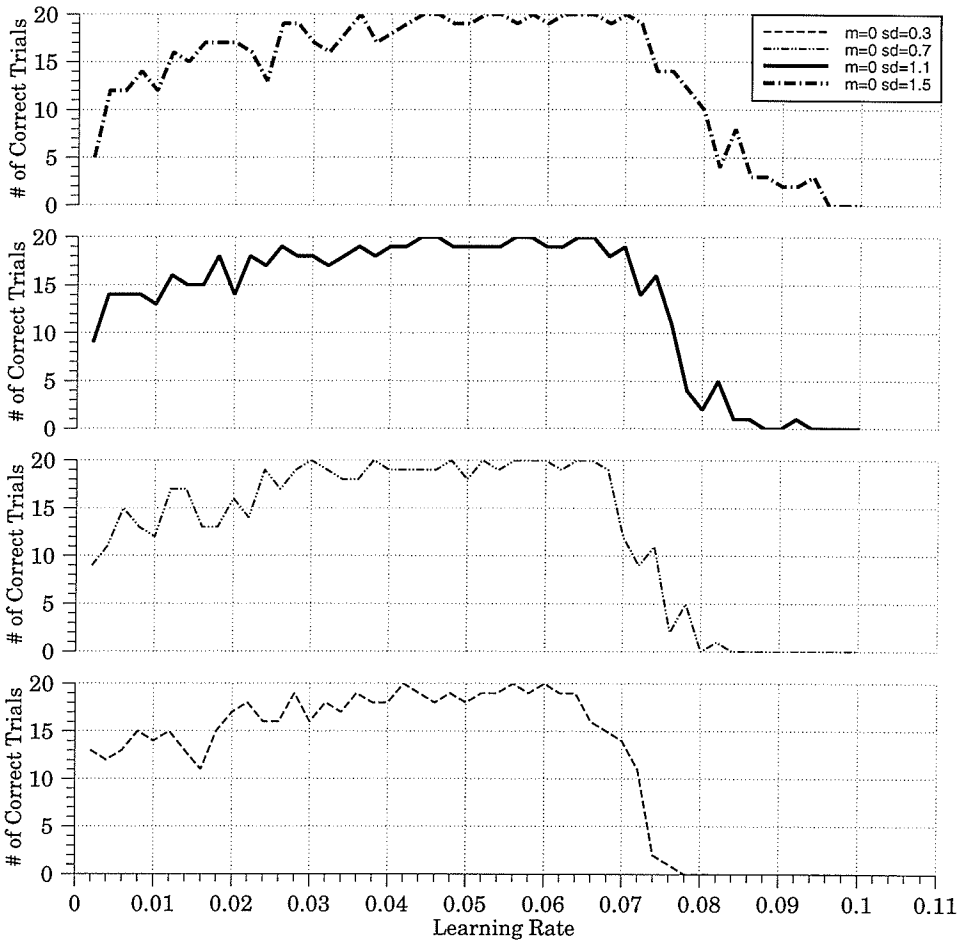

Figure 3.9: Non-ideal multiplier with noise.

the neural network decoder problem is once again simulated with the addition of a continuously changing perturbation of the multiplier slope. As before the perturbations were drawn randomly from a normal distribution and applied to the slope of the multiplier characteristic. However, for this simulation the random slope was changed before the presentation of each input pattern. The simulator was modified in such a way as to allow this perturbation effect to be included in addition to the uniform slope variations discussed in section 3.4.2. The simulations presented in Figure 3.9 represent the dynamic random perturbations with various standard deviations applied to multipliers having a small static slope variation similar to that of the bottom plot of Figure 3.6. The static variation was selected at the start of each training case and remained

unchanged for the duration of the trial.

It is quite clear from these plots that the dynamic noise present in the multipliers does not greatly effect the learning performed by the network. In all four learning cases shown the network is able to learn the decoder problem in more than 91% of the training cases over the range ε=[0.02,0.07]. This result would be expected given that the mean of the noise added to the multiplier characteristics is zero, and as a result, over an infinite number of training iterations the mean effect on the weight updates should average to zero as well. In fact the noise actually tends to compensate slightly for the static variations present in the multipliers by reducing its effects through the short term statistics of the noise.

The results of this section differ from those of the slope variations in section 3.4.2, where the network learned to compensate for the fixed variations. Here the statistics are mainly responsible for the compensation, and not the learning algorithm itself. However, these simulations show that the learning is not overly sensitive to the incremental variations in the dynamic variations. This result differs from those obtained Brion Dolenko[2] for his simulations of the backpropagation modules discussed in Chapter 2. He has found that backpropagation does not tolerate circuit noise particularly well, whereas here the noise does not adversely affect the learning. In fact, the results indicate that a small amount of noise may actually be desirable in compensating for other imperfections.

3.4.4 Effects of Multiplier Offset Variations on Learning

Another important effect that will be encountered as a consequence of the fabrication process is a zero crossing offset variation in the multiplier. This will result in the multiplier producing a non-zero output value when one or both of its inputs are zero. In order to evaluate the effects of this behaviour the sample problem was once again simulated with various degrees of offsets. The results of these simulations appear in Figure 3.10. As before, the random offsets have been selected from a gaussian distribution with a variety of standard deviations.

It is expected from actual measurements of past fabrication runs that fabrication errors will yield multiplier zero crossing offsets which are in the

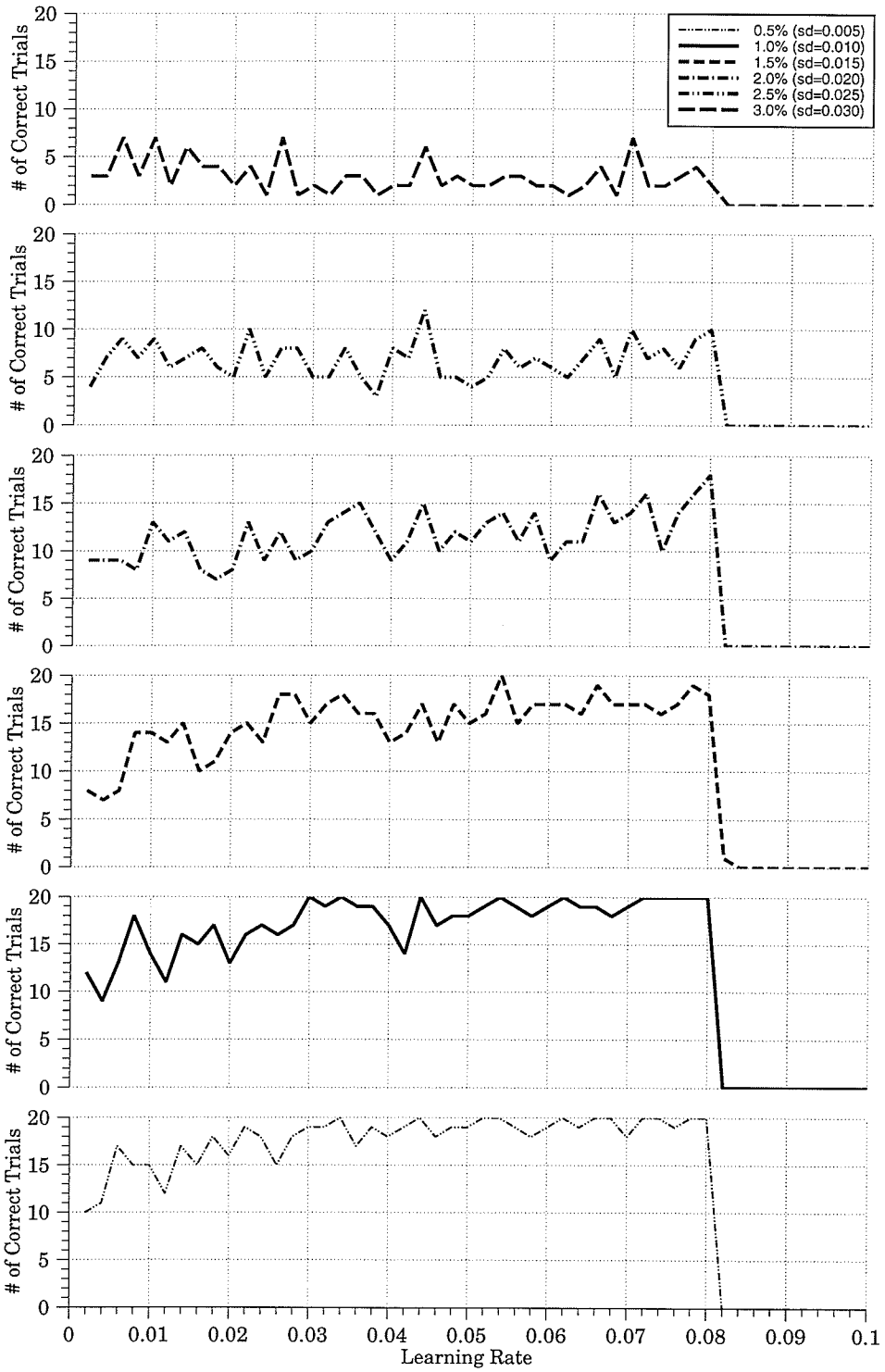

Figure 3.10: Non-ideal multiplier with offset variations.

range of +/-5% of the maximum multiplier output value. It is clear from the plots that the network is capable of tolerating a small amount of offset, but begins to fail for offsets larger than 1%. Offsets of 0–1% learn correctly on >89% of the learning trials conducted over the range ε=[0.01,0.08]. However, offsets of 1.5%, 2%, 2.5% and 3% learn correctly on only 79%, 60%, 34%, and 15% of the trials respectively. These results are clearly well below the expected level of variations for this multiplier, greatly increasing the probability that a hardware implementation will perform unsatisfactorily. This is consistent with the results obtained for backpropagation in [2]. Those simulations demonstrated that the backpropagation algorithm does not tolerate zero-crossing variations either.

This problem must be corrected before a usable hardware network can be constructed. One possible alternative to this problem would see the use of a form of weight update thresholding. Under this scheme weight values are updated only if the size of the weight change is above a predetermined threshold. If the calculated update is below the threshold the resulting update would not be applied. This approximates a true zero crossing when the multiplier product approaches what should ideally be a zero value.

3.4.5 Effects of Noisy Inputs on Learning

An actual implementation of any neural algorithm must be capable of dealing with other difficulties in addition to variations in the fabrication process. One of the most common is noise in the system, caused by noise in the input signals. In order to evaluate the effects that noisy input signals will have on the learning process, various amounts of gaussian random noise were applied to each of the input patterns before presenting them to the network. The results of these simulations appear in Figure 3.11. A variety of standard deviations were selected for the random noise, ranging from 0.1 for the bottom plot to 0.4 for the top plot. It is clear that for deviations up to 0.3 the learning is not adversely effected by the noisy inputs. Learning begins to break down for deviations of 0.4 and above. In practical applications this demonstrates that the learning is extremely robust in the presence of a significant quantity of input noise. In actual practice the amount of input noise present will be well below 10% of the input range (bottom plot). Therefore this effect will not

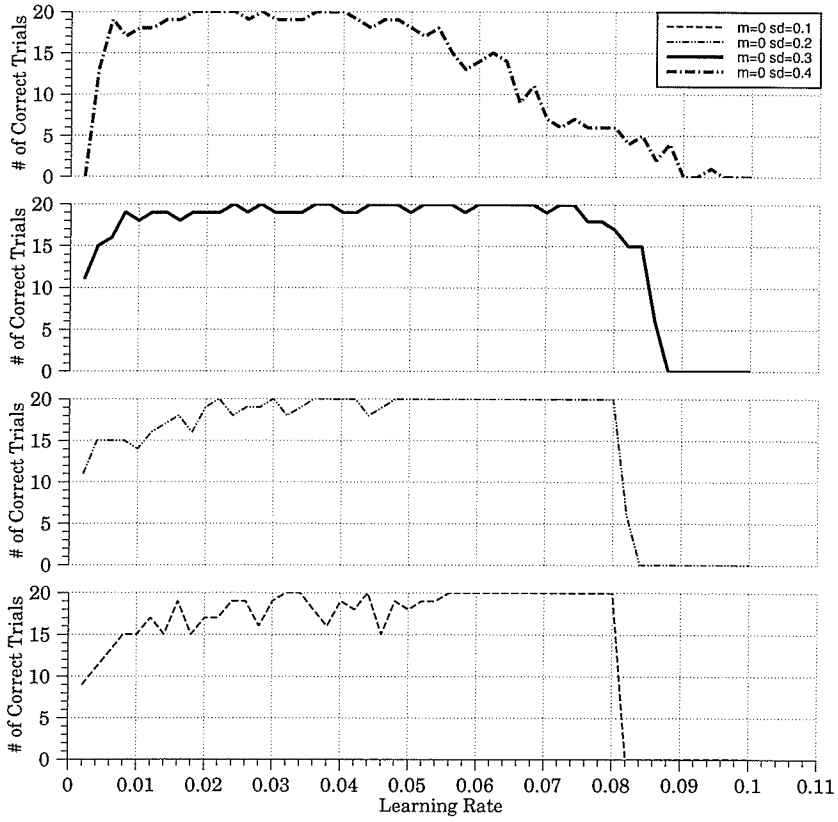

Figure 3.11: Effectiveness of learning with noisy inputs.

hinder the learning process to any significant extent. In fact, the noisy inputs should improve the performance of a network since the small variations in the input signals result in the dynamic creation of additional training patterns. The noise essentially expands the training set as a natural consequence of its presence. The additional patterns will allow the network to generalize better after learning is complete, compared to an ideal network which was only trained on the ideal input patterns.

3.5 Summary

The work presented in this chapter has examined the competitive learning algorithm and its susceptibility to analog CMOS hardware implementation difficulties. The basic theory of competitive learning has been discussed with an eventual focus on soft competitive learning. Following this discussion, a variety of hardware fabrication problems were outlined and the effects of

these variations on the competitive learning were investigated.

Investigations showed that competitive learning is able to tolerate most anticipated hardware variations except zero-crossing offsets in the Gilbert multipliers. Apart from this effect it was found that the use of a tanh(•) style multiplier, fixed multiplier slope variations, and noisy slope variations were tolerated very well. In addition it was observed that noise present in the network inputs do not effect the learning process to any significant degree. Thus an actual hardware implementation of competitive learning would be expected to operate correctly, as long as multiplier offsets could be controlled.

CHAPTER 4

Conclusions and Future Work

This thesis has given a brief overview of neural network theory, and has examined two specific neural learning algorithms in relation to their suitability for fabrication in analog hardware. The rationale for hardware implementations in general, and analog versions specifically, has been presented. It has been stated that hardware implementations which include implementation of the learning algorithm will significantly reduce the learning times of a network, compared to simulation on serial computers. As well, it was reasoned that analog implementations provide the necessary computational power needed for these algorithms while improving integration density when compared to equivalent digital techniques.

A learning algorithm for the extraction of spatially coherent information in an input stream has been described. A fully custom analog version of the CBUL algorithm was then presented. This included both the design of circuitry for the backpropagation of errors in the synapses, and for mutual information calculations in the neuron. It was shown that it is possible to implement a complex algorithm such as this, in analog hardware, using a small collection of circuit elements.

Finally, a number of simulations were presented which examined the effect which expected fabrication variation will have on the construction of a hardware version of competitive learning. It was clearly shown that the competitive learning algorithm was robust in the presence of most variations but did suffer unacceptably from multiplier zero-crossing offsets. As well, the results obtained for this unsupervised algorithm were compared with those

obtained for the backpropagation algorithm. The results observed were quite similar with the exception that competitive learning was more tolerant of noise in the multiplier characteristic than backpropagation.

4.1 Future Work

Continued work in this area should result in the layout, fabrication, and full testing of both the mutual information neuron of chapter 2, and the competitive learning circuitry of chapter 3. As well, additional tests on the CBUL synapse, which has already be fabricated, should also be performed. Effective testing of these circuits has been hampered in the past by a testing environment which is poorly suited to the test of analog devices. The hardware tests presented in this thesis had been performed using an ASIX-II digital tester which provides only a limited ability to supply analog signals, and no facility to measure them.

Additional simulations should be conducted in order to examine the effects which non-ideal circuitry has on the network settling dynamics. One would expect that hardware variations should result in longer settling time, but actual simulations are required. As well, further simulations should be conducted in order to determine the effects of fabrication variations on modifications of the standard competitive learning presented in chapter 3. These should include examinations of networks with non-uniform lateral inhibitions between the output units, frequency sensitive learning algorithms, as well as examinations of these networks with more complex learning tasks. In addition, this work can be extended to include other unsupervised learning algorithms such as self-organizing feature maps.

REFERENCES

[1] S. Becker, *An Information-theoretic Unsupervised Learning Algorithm for Neural Networks,* Ph.D. Dissertation, Department of Computer Science, University of Toronto, 1992.

[2] B.K. Dolenko, *Performance and Hardware Compatibility of Backpropagation and Cascade Correlation Learning Algorithms*, MSc. Thesis, Department of Electrical and Computer Engineering, University of Manitoba, 1992.

[3] B.K. Dolenko and H.C. Card, "The Effects of Analog Hardware Properties on Backpropagation Networks with On-Chip Learning," *IJCNN '93*, San Francisco, CA, March 1993.

[4] B. Gilbert, "A High-Performance Monolithic Multiplier Using Active Feedback," *IEEE Journal of Solid-State Circuits*, Vol. SC-9, No. 6, December 1974.

[5] D.O. Hebb, *The Organization of Behavior*, Wiley, New York, 1949.

[6] J. Hertz, A. Krogh, R. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley Publishing Co., 1991.

[7] G.E. Hinton and S. Becker, "An Unsupervised Learning Procedure that Discovers Surfaces in Random-dot Stereograms," *Proceedings of the International Joint Conference on Neural Networks*, Washington, DC, 1990.

[8] G.E. Hinton, "Connectionist Learning Procedures," *Artificial Intelligence*, Vol. 40, pp 185-234, 1989.

[9] M. Holler, S. Tam, H. Castro, R. Benson, "An Electrically Trainable Artificial Neural Network (ETANN) with 10240 "Floating Gate" Synapses," *International Conference on Neural Networks*, 1989.

[10] Y. Le Cun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, L.D. Jackel, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation*, Vol. 1, pp 541–551, 1989.

[11] R. Linsker, "Self-Organization in a Perceptual Network," *IEEE Computer*, March 1988.

[12] C. Mead, *Analog VLSI and Neural Systems*, Addison-Wesley, 1989.

[13] D.E. Rumelhart, G.E. Hinton, R.J. Williams, "Learning representations by back-propagating errors," *Nature*, 323:533-536, 1986.

[14] C. Schneider, *Analog CMOS Circuits for Artificial Neural Networks*, Ph.D. Dissertation, Department of Electrical and Computer Engineering, University of Manitoba, 1991.

[15] D. Van Camp, T. Plate, G.E. Hinton, *The Xerion Neural Network Simulator,* Department of Computer Science, University of Toronto, 1991.