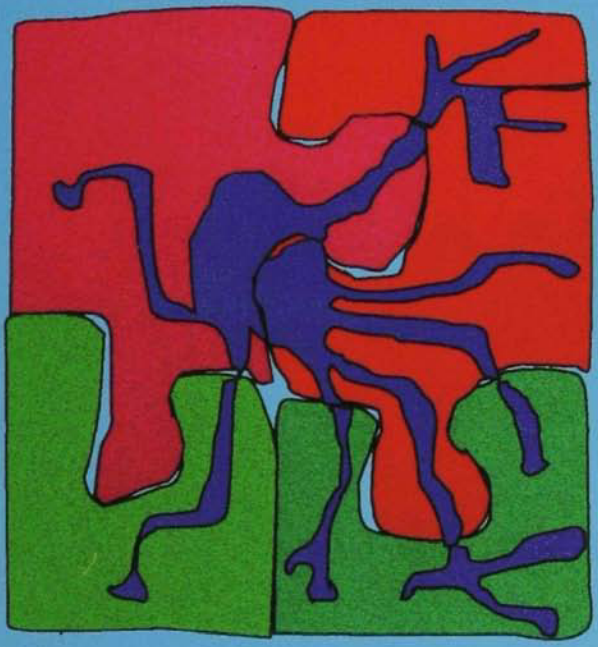
Progress in Neural Processing • 9

RAM-Based Neural Networks

Editor

James Austin



World Scientific

RAM-Based Neural Networks

PROGRESS IN NEURAL PROCESSING

Series Advisors

Alan Murray *(University of Edinburgh)*
Lionel Tarassenko *(University of Oxford)*
Andreas S. Weigend *(Leonard N. Stern School of Business, New York University)*

Vol. 1: Neural Networks: The Statistical Mechanics Perspective
Eds. Jong-Hoon Oh, Sungzoon Cho & Chulan Kwon

Vol. 2: Neural Networks in Financial Engineering
Eds. Apostolos-Paul Refenes, Yaser Abu-Mostafa, John Moody & Andreas Weigend

Vol. 3: Parallel Implementation of Backpropagation Neural Networks on Transputers: A Study of Training Set Parallelism
by S. Saratchandran, N. Sundararajan & S.-K. Foo

Vol. 4: Analogue Imprecision in MLP Training
by Peter J. Edwards & Alan F. Murray

Vol. 5: Applications of Neural Networks in Environment, Energy, and Health
Eds. Paul E. Keller, Sherif Hashem, Lars J. Kangas & Richard T. Kouzes

Vol. 6: Neural Modeling of Brain and Cognitive Disorders
Eds. James A. Reggia, Eytan Ruppin & Rita Sloan Berndt

Vol. 7: Decision Technologies for Financial Engineering
Eds. Andreas S. Weigend, Yaser Abu-Mostafa & A.-Paul N. Refenes

Vol. 8: Neural Networks: Best Practice in Europe
Eds. Bert Kappen & Stan Gielen

Progress in Neural Processing • 9

RAM-Based Neural Networks

Editor

James Austin
University of York

World Scientific
Singapore • New Jersey • London • Hong Kong

Published by

World Scientific Publishing Co. Pte. Ltd.

P O Box 128, Farrer Road, Singapore 912805

USA office: Suite 1B, 1060 Main Street, River Edge, NJ 07661

UK office: 57 Shelton Street, Covent Garden, London WC2H 9HE

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

RAM-BASED NEURAL NETWORKS

ISBN 981-02-3253-5

Printed in Singapore.

Preface

This book aims to show the state of the art in RAM based networks and to introduce the reader to this mature and developing area of artificial neural networks. It is based on presentations given at the Weightless Neural Network Workshop , Canterbury, in 1995. The authors of the chapters represent the majority of active researchers in RAM based networks through out the world. While some of the papers contain in-depth research results which may only be of interest to specialist workers undertaking research in RAM based networks, many chapters contain introductory material that describe the RAM based methods at a level suitable for workers with a basic knowledge of neural networks.

The area that the book covers is popularly known by at least three names, these being RAM based networks, weightless networks and N tuple networks, and can be used interchangeably. Other terms can that can relate to some forms of the networks in this book are Sigma-Pi networks and Binary Networks. The reason for choosing RAM based networks as the title, is because all the networks have some form of a Random Access Memory structure (RAM) in their architecture, where a RAM is the universal storage system used in current computer systems and is the main implementation model for the networks. It also reflects the major strength of RAM based systems which is there simple implementation in dedicated and high speed hardware. The term weightless networks reflects the fact that RAM based networks can be expressed purely as a logic expression with no 'weights' in the conventional neural network sense. In this case learning is seen as determining the logic function required to map the inputs to the outputs of the network under the conditions laid down in the training set. The term N tuple networks comes from the origins of the methods given by Bledsoe and Browning and their approach to selecting inputs for the network.

RAM based networks have existed for some time. Originally termed N-Tuple methods developed by Bledsoe and Browning in 1959 and being studied for some years before loosing favour in the late 1960's. They were then picked up by Igor Aleksander in the UK in the early 1970's where there simple implementation in Random Access Memories was identified and explored. His work, lead to a number of new concepts which some of his students explored and developed. In the 1980's neural networks emerged from the dark ages of the subject and the general interest in all forms of neural networks took off, including research in RAM based systems. The area has mostly been studied in the UK with one or two notable exceptions, and is now an accepted and mature subject with a number of successful commercial applications.

In terms of theory N tuple methods have had a hard time, mainly due to the difficulty in dealing with the combinatorial explosion found in most mathematical approaches to the analysis of the methods. This, along with there unique construction has put off many researchers from studying them. However, this has not stopped others taking a more pragmatic approach to the area, showing how useful problems can be

solved and high performance systems can be implemented. In addition, the book contains many useful pointers to the formal analysis of RAM based networks.

The book is structured into three sections, each introduced by an overview of the papers in the section. Section 1 contains chapters which introduce and/or compare methods known as RAM based. Section 2 presents work that extends the basic RAM based understanding and methods. Finally Section 3 contains chapters that are mainly concerned with applications of RAM based methods. However, many of these also extend the basic techniques and show, for example, how the methods may be implemented.

I would like to thank a number of people who helped in the production of this book. In particular David Bisset and the original panel who reviewed the papers for the Weightless Neural network Conference where many of the chapters saw there first drafts. Thanks goes to Aaron Turner, who checked all the latex of the chapters, my secretary, Christine Linfoot for chasing all the Authors, and finally the members of the Advanced Computer Architecture Group for reading and helping with comments on the chapters.

Jim Austin,
York, May 1997

Contents

Preface v

Section 1: RAM-Based Methods

1 Introduction 1

1.1 RAM-Based Neural Networks, a Short History 3
J. Austin

1.2 From WISARD to MAGNUS: A Family of Weightless Virtual Neural Machines 18
I. Aleksander

1.3 A Comparative Study of GSN^f Learning Methods 31
A. C. P. L. F. De Carvalho, M. C. Fairhurst, D. L. Bisset

1.4 The Advanced Uncertain Reasoning Architecture, AURA 43
J. Austin, J. V. Kennedy, K. Lees

Section 2: Extensions to N-Tuple Theory

2 Introduction 51

2.1 Benchmarking N-Tuple Classifier with StatLog Datasets 53
M. Morciniec, R. Rohwer

2.2 Comparison of Some Methods for Processing "Grey Level" Data in Weightless Networks 61
R. J. Mitchell, J. M. Bishop, S. K. Box, J. F. Hawker

2.3 A Framework for Reasoning About RAM-Based Neural Networks for Image Analysis Applications 71
G. Howells, D. L. Bisset, M. C. Fairhurst

2.4 Cross-Validation and Information Measures for RAM-Based Neural Networks 78
T. M. Jørgensen, S. S. Christensen, C. Liisberg

2.5 A Modular Approach to Storage Capacity 89
P. J. L. Adeodato, J. G. Taylor

2.6 Good–Turing Estimation for the Frequentist N-Tuple Classifier 101
M. Morciniec, R. Rohwer

2.7 Partially Pre-calculated Weights for Backpropagation Training of RAM-Based Sigma–Pi Nets 110
R. Neville

2.8 Optimisation of RAM Nets Using Inhibition Between Classes 123
T. M. Jørgensen

2.9 A New Paradigm for RAM-Based Neural Networks 130
G. Howells, D. L. Bisset, M. C. Fairhurst

Section 3: Applications of RAM-Based Networks

3 Introduction 141

3.1 Content Analysis of Document Images Using the ADAM Associative Memory 143
S. E. M. O'Keefe, J. Austin

3.2 Texture Image Classification Using N-Tuple Coding of the Zero-Crossing Sketch 155
L. Hepplewhite, T. J. Stonham

3.3 A Compound Eye for a Simple Robotic Insect 166
J. M. Bishop, D. A. Keating, R. J. Mitchell

3.4 Extracting Directional Information for the Recognition of Fingerprints by pRAM Networks 174
T. G. Clarkson, Y. Ding

3.5 Detection of Spatial and Temporal Relations in a Two-Dimensional Scene Using a Phased Weightless Neural State Machine 186
P. Ntourntoufis, T. J. Stonham

3.6 Combining Two Boolean Neural Networks for Image Classification 193
A. C. P. L. F. De Carvalho, M. C. Fairhurst, D. L. Bisset

3.7 Detecting Danger Labels with RAM-Based Neural Networks 205
C. Linneberg, A. W. Andersen, T. M. Jørgensen, S. S. Christensen

3.8 Fast Simulation of a Binary Neural Network on a Message Passing Parallel Computer 213
T. Macek, G. Morgan, J. Austin

3.9 C-NNAP: A Dedicated Processor for Binary Neural Networks 225
J. V. Kennedy, J. Austin, R. Pack, B. Cass

Author Affiliations 239

Section 1

RAM based methods

This section introduces the reader to some of the major RAM based methods. The first paper presents an overview of the RAM based methods up to 1994. It covers the most well known methods in RAM based networks. This paper is followed by an overview of the MAGNUS system, introduced by Igor Aleksander who has been one of the major influences in the development of RAM based systems throughout the last 30 years. The paper shows how RAM based systems in the form of WISARD is related to MAGNUS, a system designed to explore possibility of systems that react in an intelligent way to sensory data. The paper by DeCarvaho, Fairhirst and Bisset describes a form of RAM learning called GSN^f which allows training of multi-layered RAM based systems, and aims to compare the various forms of the GSN methods. The final paper generally introduces the AURA RAM based network, which extends the RAM based method for use in rule based systems, which is an unusual application for neural networks but exploits the speed and flexibility of the RAM based method for this task.

This page is intentionally left blank

RAM BASED NEURAL NETWORKS, A SHORT HISTORY

J. AUSTIN
Advanced Computer Architecture Group,
Department of Computer Science,
University of York, York YO1 5DD, UK

This chapter describes the interrelationship between the different types of RAM based neural networks. From their origins in the N tuple networks of Bledsoe and Browning it describes how each network architecture differs and the basic function of each. It discusses the MRD, ADAM, AURA, PLN, pRAM, GSN and TIN architectures. As such, the chapter introduces many of the networks discussed later in the book.

1 The motivation for binary neural networks

The typical model of a neuron used in a large number of neural networks is based on the McCulloch and Pits[1] neuron which can be described by equations (1) and (2). These specify a linear weighted sum of the inputs, followed by a non-linear activation function.

$$u = \sum_{j=1}^{n} w_j x_j \tag{1}$$

$$y = \frac{1}{1 + e^{-au}} \tag{2}$$

where u is the activation of the neuron, w are the weights, x are the inputs, a controls the shape of the output sigmoid, and y is the output. When equation (2) is replaced by a Heavyside function, the neuron is called a Linear Threshold Unit (LTU).

When given the appropriate activation function and used in networks with 3 layers (MLN) they have been shown to be universal function approximators[2]. In addition, they have very good generalization abilities. However, this universality comes at a cost. To train a MLN an arbitrary complex problem requires repeated presentation of training examples, which often results in very long learning times.

The most popular approach to training an MLN is the generalized delta rule, and its derivatives[3] which requires both the forward propagation phase and a backward (back error) propagation phase. The complexity of the training algorithm has limited

its use to applications which allow enough time for training the network.

2 N tuple method

The RAM based systems were not originally designed with a consideration of the limitations of MLNs, but do provide solutions to these problems. They originate in the work of Bledsoe and Browning[4], who invented a method of pattern recognition commonly termed the N-tuple method. The basic principle behind the N tuple method is that learning to recognize an image can be thought of as building a set of logic functions that can describe the problem. The logic functions will evaluate true for all images which belong to the class that the logic function represents and evaluate false for all other classes.

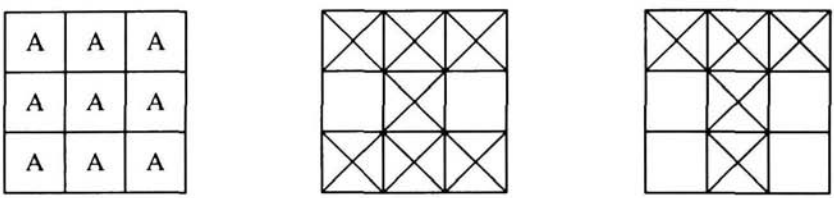


Figure 1: This is an example of the N tuple process

This shown in the simple example in Fig. 1. Each class of image has a set of logic functions that relate to it. For an unknown image, the set of logic functions that has the majority of functions which evaluate to true, indicate the class of image. The image of a T can be recognized by using the logic function;

$$R = A \cdot B \cdot C + \overline{D} \cdot E \cdot \overline{F} + G \cdot H \cdot I \tag{3}$$

The image of an I shown in Fig. 1 can be recognized by using the logic function;

$$R = A \cdot B \cdot C + \overline{D} \cdot E \cdot \overline{F} + \overline{G} \cdot H \cdot \overline{I} \tag{4}$$

To improve the generalisation ability of the N tuple method, instead of logically

ANDing the minterms of the expressions together, they are arithmetically summed to give a count of the number of terms that are true.

In the example in Fig 1 there are 3 'tuples' each of size 3. These form the minterms in the equations above. They are

Tuple 1; pixels A B C
Tuple 2; pixels D E F
Tuple 3; pixels G H I

To learn the logic functions that represent the data belonging to a given class was originally shown by Aleksander and Stonham[5]. The approach was based upon the structure shown in Fig 2. The problem involves remembering which logic terms would be needed for specific classes of image. This was most easily achieved by using a logical 1-in-N decoder followed by a set of binary storage locations for each term, and using one such unit for each tuple. The logical decoders compute all possible logical functions of the N inputs they connect to. When presented with a piece of data, the various decoders will indicate the functions required. To recognize an unknown piece of data, the same approach is used, only now the storage locations are accessed and summed.

The typical definition of a RAM node used in this book is one tuple of N inputs, followed by a logical 1 in n decoder, followed by a set of storage locations and a summing device.

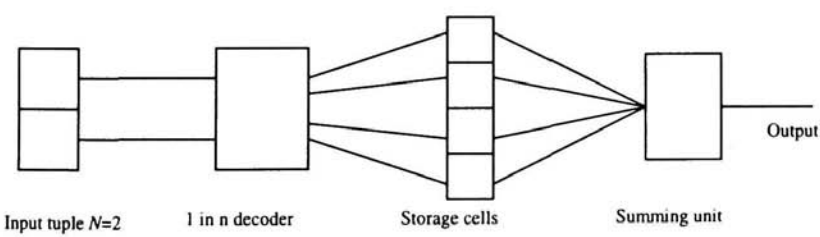


Figure 2: The basic RAM node

Input A	Input B	Output lines
0	0	1000
1	0	0100
0	1	0010
1	1	0001

Table 1: Activation table for 1 in n decoder

2.1 RAM based units compared to LTU

RAM based units are most similar to 'higher order' networks[6], which combine the input data prior to the inputs application to a system composed of LTUs. However, where higher order units combine continuous values using non-linear functions (powers etc.) RAM units combine the inputs using logical functions (AND and OR). In effect, this is the same approach, only one is continuous the other is binary.

Fig. 3 and 4 show the relation between a network of RAM units and a higher order network. Fig. 3 shows what is commonly called an N tuple network. One such network can be used to identify the similarity of an unknown image to one trained. A group of such networks, used to recognize one of a number of classes is called a Multi-RAM discriminator (MRD). Fig. 4 shows a higher order network with an equivalent structure as an N tuple network. The first layer consists of a set of decoders followed by a single layer of units with binary weights.

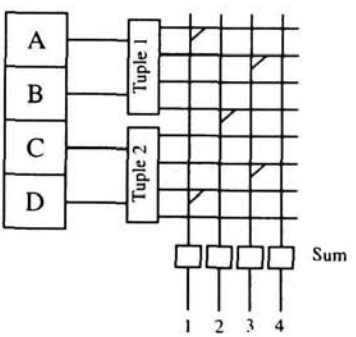


Figure 3: A single layer net with N tuple pre-processing.

Another important difference is that RAM units always take sub-samples from the image in the form of tuples. Each tuple is made up of N samples of input data, fed

to its own set of storage cells. In RAM based systems, the size of a tuple, N, is an important parameter as it effects the classification ability of the network.

The major advantage of the RAM based approach is that the decoder/storage cell combination is a random access memory (RAM). Thus allowing very simple and direct implementation in cheap and readily available components. This was shown by Aleksander et. al. [7] in the WISARD pattern recognition machine.

2.2 RAM based associative memories

The use of RAM based methods incorporating N tuple methods in associative memories has been investigated by Austin in a range of work starting with ADAM in 1986[15] to AURA in 1996 (see chapter 1.4). The work had its origins in the development of WISARD, where associativity was seen as useful in occlusion analysis in images. The use of RAM based neural networks as associative memories has been to allow high speed pattern matching on large amounts of data. The use of binary weights, and simple pre-processing has allowed the system to be implemented very efficiently in dedicated hardware as discussed in chapter 3.9. The simple training methods also allows fast storage of associations. The methods have been applied to many problems including symbolic reasoning discussed in chapter 1.4. The AURA methods now include ADAM as a subset and allow associative memory systems to be constructed for a wide variety of data types and data recall problems.

2.3 The limitations of the N tuple method

Although the basic N tuple approach was powerful, in terms of is learning speed and simple implementation. The method has a major limitation. This relates to the learning capacity of a given N tuple network. By inspection it may be obvious that a given N tuple network cannot implement all possible functions of the data inputs.

To show this a simple problem (intra-exor) problem is shown in table 2.

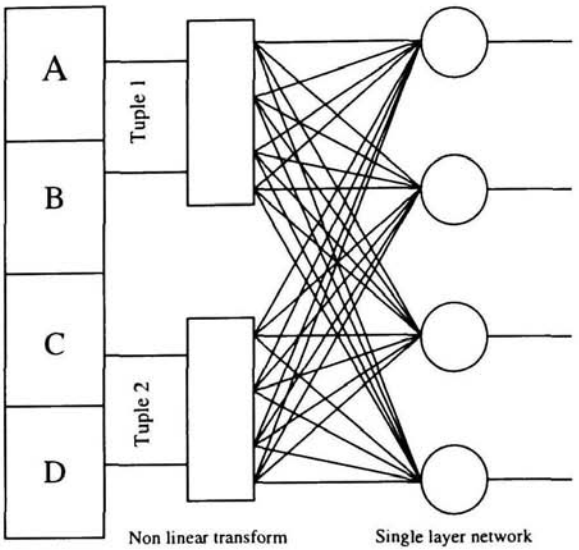


Figure 4: A single layer net with N tuple preprocessing

Inputs ABCD	Output
1010	1
0101	1
1001	0
0110	0

Table 2: The intra-exor problem

The tuple distribution given in fig. 5 shows a two tuple system that cannot solve the problem intra-exor problem shown in table 2 with a given N tuple network. However, by altering the placement of each tuple as shown in fig 6, the problem is solvable. This is unlike the EX-OR problem when used on a single layer network. The network cannot solve the problem given an arbitrary ordering of the inputs.

To solve this problem, one could used an tuple size equal to the input data size. However, this loses any generalization ability of the network, and results in RAM nodes with a very large memory requirement.

The N tuple method was used in the WISARD pattern recognition machine[7], capable of recognizing images at about 25 frames per-second, and more recently in the C-NNAP system[14].To overcome the problem given here, the training and testing method involved moving the image around whilst training and testing. This effectively ensured that most of the time the EXOR type problem did not arise (it also allowed good generalization).

For many problems, such as image processing, the intra-exor problem is not an issue, as training methods overcome the limitation. The method has been successfully applied to many problems, such as monitoring crowded underground railway platforms, face recognition[9], satellite image recognition[10] and character recognition[11], as well as the applications described in section 3.

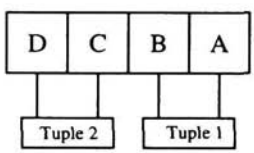


Figure 5: An arrangement of tuples that will not solve the intra-exor problem

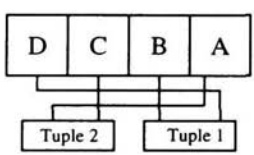


Figure 5: An arrangement of tuples that will solve the intra-exor problem

A number of extensions to the basic N tuple method have been developed. To deal with non-binary inputs Wilson[12] and Austin[13] have developed methods for taking in grey scale data and still allow the use of binary logic functions (also see chapter 2.2).

2.4 Hardware implementation of the N tuple method

The N tuple method was implemented in dedicated hardware in the form of WISARD in the early 1980's[7]. The machine was taken up commercially by Computer Recognition Systems of Woking (UK) and marketed for a number of years. The commercial system used conventional digital processor methods and achieved a recognition rate

of 12.5 frames a second for images of 512^2 images, using sparse sampling (i.e. not all pixels used).

More recently the method has been used in a parallel image processing system (C-NNAP[14]), where the method has been extended to form an associative memory[15]. This system uses Field Programmable gate arrays to implement the memory scanning and training processes.

3 Overcoming the limitations of N tuple; RAM pyramids

As the intra-exor problem has shown, the use of the N tuple method can be rather hit-or-miss. Some training examples will not be separable, while others will be.

To increase the robustness of the method the functional capacity needs to be raised whilst maintaining the generalization ability, along with the training speed, and simple hardware implementation.

To achieve this, it is important to understand how the method is capable of generalizing on unseen data. The method operates by a set membership classification process. Each image is broken up into a number of tuples. In the N tuple method, for a given set of patterns the binary patterns appearing in each tuple are recorded during training. During testing, each tuple is checked by each RAM node to see if it contains a known bit pattern, and the number of RAM units that recognize the input tuple pattern is counted and output as a recognition figure. In effect each RAM unit, which process the tuple data is looking for patterns that belong to a set of known patterns that were presented during training. The generalization comes from the way tuple patterns are allowed to be mixed between training examples. The larger the tuple size the smaller will be the generalization set size.

Unfortunately, the size of the generalization set is not set by training, (apart from the number of examples given), but by the tuple size, N. Thus, to get a good balance between classification success and generalization, requires experimentation, (see chapter 2.1 for an evaluation of this).

The problems are caused by the linear combination of the results of the RAM units. In the N tuple method these are summed. This results in the intra-exclusive OR problem given in section 2.2.

The obvious solution is to combine the outputs of the RAM units non-linearly. This can be done in two ways; use a multi-layer network of linear threshold units or use more RAM units. The former method has been called a Hybrid network[8]. This is very similar to the higher order networks, except that it uses logical functions to combine data. The approach results in a solution to the problem, but at the cost of longer training time (iterative). The latter methods are more popular, and uses the RAM based approach exclusively.

The typical form of the multi-layer RAM net (MLRN) is to combine the results of one layer of RAM units using subsequent layers. Because each RAM unit has a lim-

ited number of inputs, it is necessary to have many layers of RAMs for large images.

The typical form of a MLRN is shown in Fig.7.

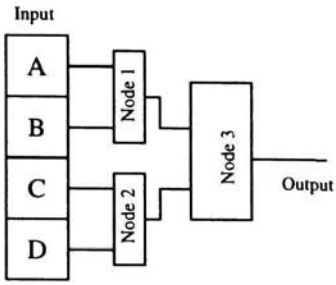


Figure 6: Typical form of a multi-layer RAM network.

These networks can implement any subset of logical combinations of the input data[16]. Thus, overcoming the intra-exclusive OR problem given earlier. However, the solution requires a new training method, for the same reason that the multi-layer networks of LTU required the introduction of the generalized delta rule (GDR). As was shown by the GDR, to perform back propagation requires soft-limited output functions on each neuron. The function implemented by RAM units is not continuously differentiable, thus the MLRNs cannot be trained using the GDR when implemented using the basic RAM node. This problem has forced a number of researchers to investigate how MLRN can be trained. Notably, the Probabilistic Logic Node (PLN)[17], the probabilistic RAM (pRAM)[18], the Goal Seeking Neuron (GSN)[19] (and chapter 1.3) and Time Integrating Neuron (TIN) networks[20] (and chapter 2.7), and their derivatives.

All these networks provide solutions to training MLRNs.

3.1 Reinforcement Learning in Multi-layer RAM nets - the PLN

Because multi-layer RAM networks cannot be trained using a back-propagation like algorithm, reinforcement learning method has been used, which does not require a direct measure of error, but just an indication that the output was incorrect. The reinforcement signal is sent to all nodes and used to determine if the pattern present on the input to the RAM unit should be saved (i.e. the logic function noted).

As the RAM node stands, reinforcement could not be used. This is because the node can only record (1) the presence of a particular binary pattern during training, or (2) the absence of the pattern. This binary representation would not work with reinforcement learning as three states are needed for the method to work, namely (1) the

pattern input caused a correct output, (2) the pattern input caused an incorrect output, (3) the pattern did not occur on the input. By allowing tri-state storage in the RAM node reinforcement learning could be used. This was implemented by Aleksander in the PLN[21]. In practice the PLN represents the information as;

1 = tuple pattern occurred and is correct,

0 = tuple pattern occurred and is incorrect,

u = pattern has not occurred.

The use of the 'u', 'don't know' state required an extension of the learning algorithm. When a 'u' state is accessed in the RAM, the output of the ram is set to 1 or 0 with a probability of 0.5. Thus any input pattern will allow the propagation of a result to the output of the network.

The approach taken in the PLN required (1) iterative learning, (2) 3 state storage locations. As a result, direct implementation in standard RAM components is not possible and training is slower. However, training is still more rapid than other approaches using LTUs.

3.2 Weighting the storage locations, N tuple RAM extensions and the PLN

The approach used by the PLN nets and by the N tuple approach only permitted each node to record the presence and absence of a particular input pattern. This approach can be softened by allowing the system to record the frequency of occurrence of the input patterns to a RAM node. This would allow the important features of a piece of data to be weighted in preference to other features.

The original N tuple method described by Bledsoe and Browning suggested this. However, because of implementation difficulties it was not used in the hardware implementations of the N tuple method. It has been shown that the approach can improve recognition accuracy[4]. To overcome the implementation problem Austin and Smith used a weighted scheme during training and converted this to a binary representation for later implementation in RAM based nodes[22]. This approach retained the improved recognition accuracy while partially maintaining the implementation efficiency.

In the case of MLRN, Mayers[21] with the *w*-PLN and Gorse and Taylor[18] with the pRAM, both realised the benefit of using a weighted scheme in MLRNs. The pRAM is described in the next section. Mayers developed the *w*-PLN which generalized 3 state storage used in the PLN to *w* states, and showed the improved recognition accuracy which resulted. The reinforcement algorithm was extended to take this into account. This approach has not been implemented, but has been used to model delay learning in invertebrates[21].

Nevill and Stonham[23] provide a description of the PLN which has the addition of

a sigmoid activation function in a *w* state PLN. In this one, the storage locations hold values with *w* states. However, after the value is addressed and read out, the value is passed through a sigmoid normalization function which gives a continuous value between 0 and 1. This value is interpreted as the probability that the unit will fire with a 1. The same reinforcement algorithm is used.

3.3 Storing probabilities in the RAM locations, the pRAM

Gorse and Taylor[24] fully extended the design of a RAM unit to a probabilistic framework. The probabilistic RAMs hold the probability of a given input pattern occurring, instead of just holding a normalized value as done in the PLN approach. By using probabilities, the likelihood of the input pattern belonging to a particular class can be calculated, rather than a boolean yes/no class membership decision.

In addition they pass the probability accessed by any input pattern to other nodes. Thus, the approach is purely probabilistic in its operation.

To implement this approach, the Kings' team have used a fully probabilistic approach, where information is passed between the nodes probabilistically. To do this they introduced a pulse coded method of communication between RAM nodes[25] which simplifies the operation and hardware implementation of the node[26]. For each cycle of operation of the pRAM, the node (1) determines if pulses are present at its inputs, (2) forms a binary pattern of the bits that are on and off, (3) accesses the memory location that is addressed by that bit pattern (4) reads out the probability, u, that the node fires from the memory location. (5) sets a one or a zero at the output of the node depending on u. Thus, over time the node will access a number of memory locations and, over time, fire with a mean rate depending on those nodes values. This is an entirely probabilistic system, including the hardware implementation, which makes it unique amongst neural network systems. The pRAM uses the reinforcement learning method to update the probabilities.

3.4 Non-reinforcement learning and probabilistic RAMs - the GSN

Another way of implementing a probabilistically based RAM node is not to use continues firing rates, but to pass the probability of the node firing using a boolean value. This method, used in the GSN[19] allows the system to work statically, i.e. at any moment the exact probability of a nodes output can be obtained. In the pRAM, to obtain the probability of the output requires an averaging of the output pulses from the neuron. In addition, the GSN does not use reinforcement learning, but a method that allows learning of an unknown pattern in one presentation. The method is reviewed in chapter 1.3. Although the GSN operates in 'one pass', it embodies a search process that is similar to a depth-first search with backtracking in AI. This search process has a large worst case search time. So, although the training set is only presented once,

each pattern can take some time to learn. Furthermore, the network may not train on one pass, as the ordering of the patterns may result in the system not finding a solution.

3.5 The crossing the divide, the sigma-pi unit

The sigma-pi unit[3] has been shown to be equivalent to a RAM based node by Gurney[20]. He shows how a RAM based node can be made equivalent to the sigma-pi units and shows how, with the addition of systems that perform a sigmoid activation on the output of a node, so that the network can use back propagation learning. He shows that, if a node is described by the following,

$$a = \frac{1}{S_m 2^n} \sum_u S_u \prod_{i=1}^{n} (1 + u_i x_i) \tag{5}$$

$$y = \sigma(a) \tag{6}$$

Where S_m is the range covered by the values stored at each addressed location, n is the number of input values (tuple size), S_u is the value held in the addressed location, u is one of the RAM addresses, and x is the input tuple pattern. The term $\prod_{i=1}^{n}(1 + u_i x_i)$ effectively activates a given address if the tuple matches the indexed address, and the term $\sum_u$ indexes all addresses in the RAM. The final equation is a sigmoid activation function σ, and the output of the neuron is given by y.

Then the operation of a RAM can be made continuous over its inputs and its outputs. As he points out, the practicality of this model is bound by the size of n (the number of bits in the input). He then proposes a stochiastic version of this model (TIN) which reduces the computational complexity. The approach is similar to that taken in the pRAM, using bit streams as inputs and outputs of the unit. However, his model incorporates a sigmoid output function which makes the node continuous. The result of this is a node that can be interpreted as a continuous system and, as Gurney shows, can be trained using a version of back error propagation. However, it is not clear if any advantage is gained from this in terms of hardware implementation or speed.

The hardware implementation of this node type has been described by Hui, Morgan, Gurney and Bolori[27]. The most recent chip implements 10240 neurons in ES2 1.5um double metal CMOS.

4 Concluding remarks

The development of the RAM based approach has come a long way since the original concept was proposed by Bledsoe and Browning in 1959. The original RAM imple-

mentation is still the simplest to implement in dedicated hardware, with theoretical processing speeds in the order of 10's of nanoseconds for both training and testing. The power of the approach has been demonstrated in the ADAM system and the grey scale extensions to the method. The speed of the method was traded off against robustness, in that all training patterns may not be learned by the system. The MLRNs overcame these problems allowing complex problems to be learned without the possibility of the network failing on any examples. The original PLN showed how reinforcement learning could be used in RAM networks, requiring only the addition of a one extra state in the nodes memory locations. Increased performance has been gained by the use of multi-state *w*-RAMs, but at the cost of higher implementational complexity. The pRAM allowed simpler hardware implementation as well as adding better classification ability and a clear biological interpretation. Both the pRAM and the PLN required the addition of iterative training, which breaks with the one shot training ability of the original N tuple method. To overcome this the GSN introduced a scheme where patterns where only presented once for training, but introduced a search process in each training cycle. This allowed on-line training and up-date to a network. Finally the TIN neuron and its derivatives clearly showed that the RAM unit could be trained using conventional methods if certain variations were added.

Four major implementations of the RAM based systems exist. The original WISARD (discrete components), the ADAM multi-processor (C-NNAP, FPGA), the pRAM (VLSI) and the Hui et al. (VLSI) implementation. All have shown that hardware implementation of the learning algorithms is feasible, and that many potential applications can benefit from the hardware implementation of these methods.

The RAM based neuron undoubtedly provides a novel approach to the design of neural network systems, which provide a range of techniques for many applications. As chapter 2.1 shows, they are capable (even in their simplest form) of equalling the performance of many other statistical and neural based methods.

5 Summary

This chapter has briefly described the work in the area of RAM based neural networks. It has shown that these neurons all have a non-liner function prior to the weights, which provides the rapid training characteristics. In addition, when the binary version of the nodes are used, the implementation of the decoder and storage is simply a logical decoder and set of bit storage locations.

References

1. W S McCulloch, W Pitts. *A logical calculus of the ideas imminent in nervous activity*. Bulletin of Mathematical Biophysics, 5:115-133, 1943
2. J M Zurada. *Introduction to Artificial Neural Systems*. West Publishing Co., 1992

3. D E Rummelhart, J L McClelland. *Parallel Distributed Processing*. MIT Press, 1986.
4. W W Bledsoe, I Browning. *Pattern Recognition and reading by machine*. In Proc. Joint Comp. Conference, pages 255-232, 1959.
5. I Aleksander, T J Stonham. *Guide to pattern recognition using random-access memories*. Computer and Digital Techniques. 2(1):29-40, 1979.
6. Yoh-Han Pao. *Adaptive Pattern Recognition and Neural Networks*. Addison-Wesley, 1989.
7. I Aleksander, W V Thomas, P A Bowden, *Wisard: A radical step forward in image recognition*. Sensor Review, pages 120-124, July 1984.
8. J Austin. *Rapid learning with a hybrid neural network*. Neural Network World, 5:531-549, 1993.
9. I Aleksander. *Wisard: A component for image understanding*. IEEE Proc., 135 Pt. E(5), 1985.
10. J Austin, S Buckle. *The Practical application of binary neural networks*. In Proc. of the UNICOM seminar on Adaptive computing and information processing, 1994.
11. R Tarling, R Rowher. *Efficient use of training data in the n-tuple recognition method*. Electronics Letters, 29(4), September 1993.
12. M J D Wilson, I Aleksander. *Adaptive windows for image-processing*. IEE Proceedings-E Computers and Digital Techniques, 132(5):233-245, 1985.
13. J Austin. *Grey Scale n tuple processing*. In Josef Kittler, editor, Lecture Notes in Computer Science:301, pages 110-120. Springer-Verlag, 1988.
14. J Austin, A Turner, S Buckle, M Brown, A Moulds, R Pack. *The cellular neural network associative processor, C-NNAP*. IEEE Computer, to be published, 1994.
15. J Austin, T J Stonham. *An associative memory for use in image recognition and occlusion analysis*. Image and Vision Computing, 5(4):251-261, Nov. 1987.
16. R Al-Alawi, T J Stonham. *A training strategy and functionality analysis of digital multi-layer neural networks*. Journal of Intelligent Systems, 2(4):53, 1992.
17. W K Kan, I Aleksander. *A Probabilistic logical neural network for associative learning*. In Proc. of IEEE 1st annual Conference on Neural Networks, pages 541-548, San Diego, 1987.
18. D Gorse, J G Taylor. *Review of the theory of pRAMs*. In NM Allinson, editor, Weightless neural network conference, pages 13-17, University of York, 1993.
19. E C D B C Filho, M C Fairhurst, D L Bisset. *Adaptive pattern recognition using the goal seeking neuron*. Pattern Recognition Letters, 12:131-138, 1991.
20. K N Gurney. *Training networks of hardware-realisable sigma-pi units*. Neural Networks, 5:289-303, 1992.
21. C E Mayers. *Delay Learning in Artificial Neural Networks*. Chapman and Hall, 1992.

22. G Smith, J Austin. *Analysing aerial photographs with ADAM*. In International Joint Conference on Neural Networks, Baltimore, USA, Hune 1992.
23. Nevill, T J Stonham. *Adaptable reward penalty for PLN*. In ICANN'94, page 631, 1992.
24. D Gorse, J G Taylor, T G Clarkson. *Extended functionality for pRAMs*. In ICANN 94, pages 705-708, 1994.
25. D Gorse, J G Taylor. *An analysis of noisy RAM and neural nets*. Physica D, 34:90-114, 1989.
26. T G Clarkson, CK Ng, Y Guan. *The pRAM: An adaptive VLSI chip*. IEEE Transactions on Neural Networks, 4(3):408-412, May 1993.
27. T Hui, P Morgan, K Gurney, H Bolori. *A cascadable 2048-neuron VLSI neural network with onboard learning*. In ICANN '92, pages 647-651, 1992.

FROM WISARD TO MAGNUS: A FAMILY OF WEIGHTLESS VIRTUAL NEURAL MACHINES

IGOR ALEKSANDER
Department of Electrical and Electronic Engineering,
Imperial College of Science Technology and Medicine, London, UK

This chapter reviews a progression of weightless systems from the WISARD single-layer pattern recogniser to recent work on MAGNUS, a state machine designed to store sensory experience in its state structure. The stress is on algorithmic effects, that is, the effect of mapping these systems into conventional processors as virtual neural machines. The chapter briefly reviews the changes from the generalisation of discriminators in WISARD to generalising RAM (G-RAM) as currently used in MAGNUS systems. This leads to the introduction to MACCON (the Machine Consciousness Toolbox) a flexible version of MAGNUS which runs on current PC operating systems.

1 Introduction

There is no need to rehearse the history of the various steps that led to the WISARD and MAGNUS systems as the full story may be found elsewhere[1]. The point stressed here is that the most important application of weightless systems may still lie in the future: the cognitive behaviour if neural state machines, rather than the need for ever improving pattern recognition systems which is a central driving force from the past. Generalisation is at the centre of concern in pattern recognition while control and partitioning of state space is important in recursive systems which, in the case of weightless systems, are neural state machines. An interesting trend in the technology of such developments is the optimality or otherwise of hardware construction. While the original WISARD built by Bruce Wilkie[2] was made entirely in hardware, the subsequent commercial version[3] made use of a standard architecture with a wide bus. More recent developments in programming technology have brought this work entirely into the algorithmic domain, the original MAGNUS being a C++ program under UNIX, which in its latest MACCON format (Machine Consciousness Toolbox) is in C++ under Windows95 or NT. It is therefore the algorithmic nature of the weightless paradigm which is stressed in this chapter.

Does this mean that the learning advantages of neural systems are lost to the algorithmic methodology? Not at all. It is felt that the important change is that from real to virtual machine, the virtual machine retains all the important properties of cellularity and notional parallelism which go with the earlier weightless systems, but imports the extraordinary property of virtuality, which in the case of MACCON, turns a laptop

into a neural network with considerable cognitive power and flexibility. This leads to a level of rapid system prototyping which is currently inconceivable in special-purpose hardware.

The other trend is the exploitation of the power of state space in neural state machines. It will be argued that the associative powers of a single state coupled with the links achieved through state transitions provides an easily accessible knowledge store which takes neural systems beyond the mere labelling of patterns into schemes that can become artificially conscious of the characteristics of environments sensed through inputs. The controversial words 'become artificially conscious' are chosen with considerable care: they imply creating a scheme which has a memory of the past, a perception of the present and a control model which allows the net to understand its environment in a predictive way.

2 Discriminator algorithms

The WISARD contains several discriminators, each containing k RAMs with n address inputs such that the totality of *kn* inputs are connected to a W-point binary input pattern interface on a one-to-one basis when $kn = W$, an oversampled manner when $kn > W$ and an undersampled manner when $kn < W$. Starting with all RAM locations set to 0, the system is trained to store 1s having assigned each discriminator to a given class. Only one the appropriate discriminator is trained for each training pattern. An unknown pattern is assigned to the class of the discriminator which responds with the highest number of 1s. In algorithmic form this is well known to be identical to the n-tuple recognition algorithm of Bledsoe and Browning developed in 1959[4].

It has been shown[1] that to a first approximation the response of each discriminator is of the order of $Rj=(Aj)^n$ where Aj is the overlap similarity (that is [W- Hamming Distance]/W) between the unknown pattern and that most similar to it in the training set for discriminator j. This means that the WISARD is merely an assessor of Hamming distance between the unknown and all the patterns in the training set. In the first instance the correctness of that assessment is independent of the value of n. However a key property of such systems is that they provide a confidence level C which is defined as

$$C = \frac{(Rjmax - Rknext)}{Rjmax}$$

where $Rjmax$ is the response of the leading discriminator and $Rknext$ that of the next strongest one. Clearly n features in this as $C = 1 - [Ajnext - Akmax]^n$ where $Ajmax$ is the similarity leading to the response $Rjmax$ etc..

To summarise the WISARD algorithm therefore, an unknown pattern is assigned

to the class to which belongs the nearest Hamming distance pattern of the training set, with a confidence level which depends on the ratio of the highest class to the next highest one taken to the power of *n*. The advantages of this are that a high degree of discrimination can be obtained with a high n and that a training set for one class can contain a variety of differing patterns giving correct recognition if the unknown is nearer to any element of that set than that of any other set. The disadvantages are that the total memory cost of the system are *Dk(2)n* [where *D* is the number of discriminators] introducing a penalty for high levels of discrimination both the discrimination and the cost being exponential functions of *n*.

3 Node generalisation algorithm

In recent work the generalisation of weightless systems has been ensured through a storage of all the address patterns of a node and the application of a simple nearest-neighbour algorithm to any unknown pattern. This is known as the VG-RAM (Virtual Generalising RAM). The algorithm goes as follows.

- The training set for a particular VG-RAM consists of patterns *T=(t1, t2,..tm)*;
- Given a set *V* of classes, training consists of storing each *tj* together with its appropriate class *vj* from *V=(v1,v2..)* as a training set pair *(tj,vj)*.
- Given an unknown pattern, the nearest (Hamming) pattern[1] from *T* is found by search (say it is *tm*)
- The output of that node is assigned the class *vm* simply by reading the second half of the pair *(tm,vm)*.
- If a clear nearest neighbour cannot be found (i.e. there is a conflict between the classes of several nearest neighbours) the node is given a randomly selected output from *V*.

It may be shown that given a single layer of VG-RAMs each sampling an n-tuple, the performance of this as a recogniser is identical to that of the WISARD algorithm. The way that an overall response is computed is to add the labels within each class given in response to an unknown input. Each class tally becomes equivalent to a discriminator response (to a first approximation) and follows the $R=(A)^n$ rule (*R* now being the tally for the class). The advantages are that the storage penalty is now only *Tkn*

1. It has often been stressed in the literature that any suitable similarity measure could be introduced. Hamming distance is nevertheless convenient, and is assumed in the totality of this chapter.

where T is the total number of training patterns (irrespective of class), while the adjustment of discrimination is still an exponential function of n. There is however a time penalty, in the sense that if the average number of training patterns per class is Tc, then, in very broad terms, the VG-RAM algorithm can be expected to run Tc times longer than the WISARD one. The background to these considerations is given in[1].

In summary, the VG-RAM is favoured because of its reasonable storage requirements with respect to WISARD-like methods in pattern recognition. It will be shown next, that the VG-RAM is also well suited to being the building brick of neural state machines such as the MAGNUS structure.

4 The neural state machine (or GNU- General Neural Unit)

4.1 Structure

The real promise for neural networks, particularly weightless ones, may be in areas where knowledge storage is required - that is, recursive systems. The simplest of these is the single neural state machine, which may be described as follows in algorithmic terms. A state at time t is defined as

$$S(t) = s1(t), s2(t) \ldots sw(t)$$

sj(t) being the *j*th binary state variable at time *t* there being *w* state variables altogether.

The system also has an input at time *t* defined as

$$I(t) = (i1(t), i2(t)) \ldots iv(t)$$

ij(t) being the *i*th binary input variable of a *v*-variable input pattern.

Each *sj(t)* is computed as follows

$$sj(t) = G[i1(t-1), i2(t-1), \ldots in(t-1), \ldots s1(t-1), s2(t-1), \ldots sf(t-1)]$$

where G is computed using the G-RAM generalisation function discussed in sec-

tion 3, and the notation is used to indicate that each state variable is computed on the basis of n random samples from the input and f random samples from the state variables themselves.

4.2 Training

Clearly G can be accomplished only once some training has been done. Several general training algorithms may be deployed. A supervised algorithm will be mentioned followed by three unsupervised ones.

Supervised training (forced transition)

This is a rather obvious form of training where some target $S(t)$ exists for a given $S(t-1)$ and $I(t-1)$. Knowledge of these vectors is sufficient for the training of the state variables to take place according to the connections in 4.1. More substantial is the notion that given any $S(t)$, $S(t-1)$ and $I(t-1)$ three training procedures may be defined: transfer mode, attractor creation and input habituation.

Transfer mode

One step of training is applied to link $S(t)$ to $S(t-1)$ and $I(t-1)$. Note that the generalisation of the node will do more than just create the trained transition. Say that there are T training triplets

$$S1(t-1), I1(t-1) \rightarrow S1(t)$$
$$S2(t-1), I2(t-1) \rightarrow S2(t)$$
$$\vdots$$
$$ST(t-1), IT(t-1) \rightarrow ST(t)$$

Then, given some previously unseen $S(t-1)$ and $I(t-1)$, and given a high connectivity, the computed $S(t)$ will be that which has been learned for the vector $Sj(t-1)$, $Ij(t-1)$ which is nearest in Hamming distance, that is, $Sj(t)$. As the connectivity is reduced, it becomes possible to output a mixture of learned state variables (and only a mixture of learned ones). Braga has studied this problem[5] and its effect on the retrievability of information in state machines.

The main property of transfer training is that it can create an inner state irrespective of the current inner state. That is, assuming high connectivity, if $Sj(t-1)$, $Ij(t-1)$ ($Sj(t)$ is the jth training step, then $Su(t-1)$, $Ij'(t-1)$ will lead to a state $Sj(t)$ if $Ij'(t-1)$ is closer in Hamming Distance to $Ij(t-1)$ and $Su(t-1)$ is arbi-

trary in the sense that it is roughly orthogonal to any trained state. Of course, departures from the ideals assumed in this result lead to noise in the generation of the desired state.

Attractor creation

This consists of extending training step j (that is, $Sj(t-1), Ij(t-1)$ $(Sj(t))$ as follows

$$Sj(t), Ij(t) \rightarrow Sj(t+1)$$

where $Sj(t) = Sj(t+1)$ and $Ij(t) = Ij(t-1)$.

This 'closes the loop' on state $Sj(t)$. The generalisation according to the G-RAM algorithm ensures that for an arbitrary state Su which is not part of training where attractor creation has been applied, the system (given high connectivity) will skip into the nearest Sj which is part of the training. While lower connectivities will lead to an entry into this state in steps[5], they will also create the possibility of entering false attractors or cycles. Overall, training with attractor creation enhances the possibility of finding the state appropriate to a particular input.

Input habituation

This is a further extension of the training step j beyond the Attractor creation level to

$$Sj(t+1), \Phi(t+1) \rightarrow Sj(t+2)$$

where $Sj(t+1) = Sj(t+2)$ and $\Phi(t+1)$ is an arbitrary noise signal.

In some habituation experiments this step is repeated several times with arbitrary noise signals. The overall effect is to make the attractor less dependent on input. These three methods will be illustrated further in section 6.

Unsupervised learning

Three methods of unsupervised learning come to mind: localised region growing, random state selection and iconic. Our work has mainly centred on the last of these three. The algorithmic issue is how does one choose the internal state to accompany an ex-

ternal input or series of inputs. Region growing consists of defining neighbourhoods of each state variable in terms of other state variables. The two- or three- dimensional methods of Teuvo Kohonen are well known as the pioneering examples of this method[6]. The end result is localised firing, which through its position indicates a pattern class. On the other hand, random state selection leads to a distributed representation through an arbitrary selection of $Sj(t)$ in any of the three methods of training mentioned in the last subsection. This means that attractors may be created which are fewer in number than the number of inputs in the training set. In addition to conventional pattern labelling such systems can serve to create inner state cycles which recognise sequences of inputs. To do this, the instructor merely needs to decide when the internal state needs to be mapped with an input back to a previous state in the sequence.

4.3 Iconic learning

Most of the recent work in weightless systems at Imperial College has focused on cognitive modelling using neural state machines. The key requirement here is that the internal state structure should be a recallable representation of experience. The methodology used in this context is called Iconic Learning which was first analysed in 1991[7] and discussed more fully in terms of its cognitive implications in 1993[8] and as a contribution to neural models of consciousness in 1996[9]. It is also reviewed in[1]. Its algorithmic nature is simply stated.

Taking the general form of training seen earlier,

$$Sj(t-1), Ij(t-1) \rightarrow Sj(t)$$

a relationship is struck between $Sj(t)$ and $Ij(t - 1)$ so that the binary variables of $Sj(t)$ sample the variables of $Ij(t - 1)$. This is written as:

$$Sj(t) = \mu(Ij(t-1))$$

If the input is considered to be the result of a sensory measurement, then the states created by iconic training retain a somatotopic representation of the input. Some examples of this action will be given in section 6 of this chapter.

5 The MAGNUS concept

5.1 Raison D' Être

The MAGNUS (Multi-Automata General Neural Unit Structure) concept was a development of the GNU based on the realisation that multiple state machine architectures are important in achieving cognitive tasks. For example, the representation of duration, sequentiality and output sequence generation would require more than one state machine. The system was designed to be constructible in as flexible way as possible. An example of progressively structured architectures is given in the next section.

The MAGNUS allows the user to create a virtual machine designed to study various aspects of cognition. One of its designers, Richard Evans has done this to some effect by creating a 'virtual robot' which finds its way around a virtual 'kitchen world' [10]. This system included a way for the output of the MAGNUS to move its window around the kitchen world image altering its size if required. Evans' results include object finding and object naming in the kitchen world.

5.2 Machine Consciousness Toolbox (MACCON)

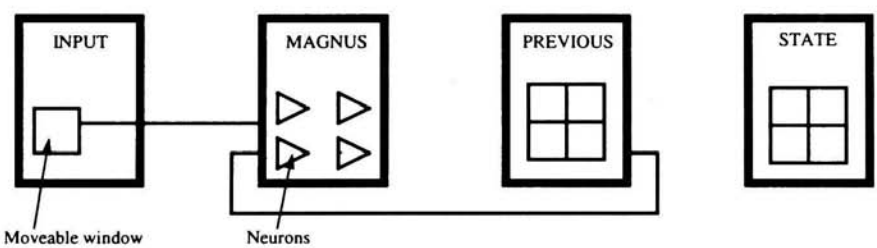


Figure 1. A simple MACCON configuration.

As explained in 1 above and developed at some length in[9], the words "Artificial Consciousness" point to a general programme of research into a variety of cognitive matters which, when taken together, contribute to an overall statement of the way in which a neural organism could become 'conscious' of its sensory world and act on it. MACCON is being developed as a prototyping set of tools based very much on the original MAGNUS architecture, but capable of being run under Windows95 or NT. This is the first reference to this virtual system which is still in a state of development in conjunction with Barry Dunmall of the Novel Technical Solutions (NTS) company.

MACCON starts by allowing the user to define a sensory world. This consists either one or more image frames into which stored images can be loaded or fields over

which a window may be positioned, the content of the window being a sensory input to the neural system. Next, one or more neural state machines can be configured. A simple example is shown in fig. 1.

The number of neurons in the block called MAGNUS (which is a neural state machine) and the number of inputs per neuron can be selected at will during the configuration phase. The display shows the previous state which, together with the input, form the interface sampled by the neurons in the net to produce the new state at the arrival of a clock pulse (controlled by the user). On clicking a 'run' button the content of the STATE window first are shifted to the PREVIOUS state window before the new state is computed. An interesting feature of the system is that it can cope with colour images as discussed in the Appendix. An example of the use of this simple configuration follows below to illustrate some of the concepts introduced earlier in this chapter.

5.3 An example of the use of MACCON

A small network of 30×30 neurons with 64 inputs each (32 feedback and 32 from the input) was set up with the MACCON in colour mode and in the configuration shown in Fig. 1. The training set consisted of only two images as shown in fig. 2

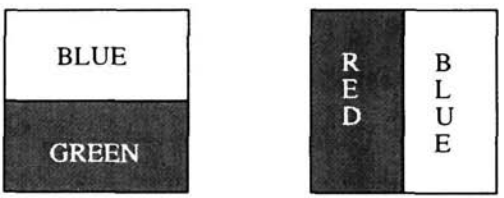


Figure 2. Two training patterns.

These consist of a red vertical bar on a blue background and a green horizontal bar also on a blue background. These patterns have been chosen because they have simple overlap regions which facilitates a theoretical discussion. The object of the tests is to elucidate the difference between the three types of iconic transfer: transfer mode, attractor creation and input habituation.

Three tests are involved:

a. **Recognition**: where the accuracy of the inner representation as a function of time is measured;

b. **Memory**: where the ability to retain the recognised state even when the input is replaced by an arbitrary noisy pattern is measured, and

c. **Attention**: where the ability to switch from one inner representation to another by a (brief) presentation of the other input (before applying a noisy pattern) is measured.

All testing is done on the training set.

Recognition	$t=0$	$t=1$	$t=2$	...	$t=10$
Transfer Mode (% OK)	50	97	95	...	97
Attractor Ctn. (% OK)	50	98	100	...	100
Input Habtn (% OK)	50	98	100	...	100
Memory					
Transfer Mode (% OK)	97	50	50	...	50
Attractor Ctn. (% OK)	100	90	87	...	71
Input Habtn. (% OK)	100	98	97	...	95
Attention					
Transfer Mode (% OK)	N/A	N/A	N/A	...	N/A
Attractor Ctn. (% OK)	100/0	22/66	25/67	...	23/68
Input Habtn. (% OK)	100/0	5.5/94	0/100	...	0/100

In the recognition tests above, the first pattern is an arbitrary noise state. In the transfer mode the resulting error is due to several factors. First, a particular neuron could be atypically connected. This means that the connections from the input are so unevenly connected that the neuron does not discriminate between elements of the training set. A particular input connection could be in the blue area common to the two training patterns (probability=1/4), or it could be insensitive to both of the colours used in the two training patterns (probability = (7/8)×(7/8)). The total probability of a single neuron input being atypically connected is

$$\frac{1}{4} + \frac{3}{4}\left(\frac{7}{8} \times \frac{7}{8}\right) = 0.842$$

Consequently, the probability of all inputs (from the non-feedback part of the overall input to a neuron) being atypically connected is

$$(0.842)^{32} = 0.00407$$

The rest of the error may be accounted for through the possibility that in testing, the arbitrary feedback state may be closer to one of the incorrect training patterns. This is made relatively easy through the fact that the Hamming distance on the input side is at most 32/4. Bearing in mind that it is most likely that $^1/_4$ of the inputs will be in the common blue area, the Hamming distance is more likely to have a maximum of 24/4, that is, 6. This additional error is hard to calculate as it involves taking into account all the possible distributions of inputs into the four quadrants of the input space. The results indicate, however, that this is a greater contribution than that likely from atypical connections.

The object of these tests is to show that feedback and attractor creation removes the error. This is borne out by both the attractor creation and input habituation results. This is due to the fact that, while the error calculated above is still present in the next step, the second step removes the ambiguity in the feedback state. So only the atypicality error is left, reduced (because it applies to input and feedback variables) to (0.0047×0.0047), that is 0.0000166 which is too small to be noticed.

In the **memory** tests, the transfer mode would not be expected to have any memory, and the results confirm a return to an arbitrary state as soon as the input is applied. In fact the arbitrary state is not purely random, but a mixture of the two trained patterns as might be expected from the training algorithm. The attractor creation, on the other hand, demonstrates something akin to short term memory as might be expected from the dependence of the output on the previous state which is deteriorating due to the lack of control from the input. The result shows a situation where the input is in a false minimum which still retains some of the memory of the original state. False minima also exist which are mixtures of the two trained states, that is, there is no long term memory. Long term memory is clearly indicated in the case of input habituation, the error being due to the same causes as in input transfer, but this time generated by the feedback state rather than the input.

Attention tests indicate that the system in one memory state may be switched into another by a single application of the alternative input. The results are displayed to show the percentage of each training state present in the current state. Initially a perfect memory state is present. The alternative state is presented for one step at the input and the response shown under t=1. The result at t=2 is with the input replaced by an arbitrary noise state. As expected, the attractor creation training remains influenced by an unrecognised input, and the full alternative state is not achieved. The input habituated system, on the other hand, achieves a full switch to the appropriate state. Further tests show that starting in an arbitrary state, the habituated system falls fully into one of the two attractors showing the usefulness of this form of training: the training ex-

perience is retrievable with high accuracy through short exposures at the input. Distorted inputs too cause falls into nearest perfectly remembered states.

4 Conclusions

In much of the weightless neural systems literature it is often stated that a major advantage of the technique is its ease of implementation in hardware. In this chapter it has been argued that an appreciation of the algorithms used by weightless methods leads to an easy creation of virtual machines which take neural systems forward into cognitive computation. Starting with the Bledsoe and Browning n-tuple algorithm (which, in hardware became the WISARD), it has been recalled that the RAM function may be augmented (G-RAM) by a node generalisation algorithm, through a spread of stored training content to addresses close in Hamming distance to the training addresses. The neural state machine was introduced to indicate the way in which spreading helps in the creation of various state space structures. MAGNUS and its portable version MACCON are virtual machines in which neural state machines may be interconnected to study hypotheses related to cognition. Of course, if cognitive behaviour is required in a system, then such machines may be used as cognitive engines.

APPENDIX: The Colour Neuron

In colour mode, each pattern interface pixel can be in one of a set **C** of eight 'colours':

C = {Black, White, Red, Yellow, Green, Cyan, Indigo, Blue}

Any input of the neuron is 'dedicated' to one of these colours, that is, if and only if the dedication of the input matches the colour of the pixel, the input is set to 1. So a neuron connected to some monochromatic field, say red, is expected to have 1/8 of its inputs set to 1. A change of colour in the monochromatic field would lead to a different set of 1/8 ones to be at 1, the set being disjoint from the first one. It is still possible to calculate the Hamming distance characteristics for any two colour images. Say that the two images have different colours in P of its pixels. As approximately 1/8 of P pixels would be coded as 1 in the first image, and a different 1/8 in the second image, the effective Hamming distance H between the two is most likely to be $(1/8 + 1/8)P$, that is,

$$H \approx \frac{P}{4}$$

Training proceeds by associating the input vector with one of the colour message.

Clashes can still occur if an attempt is made to associate more than one colour with the same input vector. In this case the output message stored is 'u' but, in contrast with the 0/1 case, 'u' causes one of the eight colours to be generated at random (where in the 0/1 case it is 0 and 1 which are arbitrarily selected).

References

1. I. Aleksander. and H.B. Morton, Introduction to neural computing, International Thomson Press, 1995.
2. B. A. Wilkie. A Stand-alone, High Resolution, Adaptive Pattern Recognition System. PhD Thesis, Brunel University, 1983.
3. I. Aleksander, W. Thomas, P. Bowden, P. WISARD, a radical new step forward in image recognition, Sensor Review, 120-4, 1984.
4. W. V. Bledsoe, I, Browning, Pattern recognition and reading by machine. Proc. Eastern Joint Computer Conf. pp225 -232, 1959
5. A. P. Braga, Interference and Sequential Learning in Recurrent Networks with Generalising Boolean Nodes. Int. Workshop on Artificial Neural Networks, Malaga, 1995.
6. T. Kohonen, The Self-Organising Map. Proc. IEEE, 78/9:1464-1480, 1990.
7. I. Aleksander, and H. B. Morton, A General Neural Unit: Retrievability. IEE Electronics Letters, 27, 1776-8
8. I. Aleksander, H. B. Morton, Neurons and Symbols, the Stuff that Mind is Made of. London: Chapman and Hall, 1993.
9. I. Aleksander, Impossible Minds: My Neurons, My Consciousness. London: Imperial College Press, 1996.
10. R. G. Evans, A Neural Architecture for a Visual Exploratory System. PhD. Thesis, Imperial College, London, 1996.

A COMPARATIVE STUDY OF GSN^f LEARNING METHODS

A. C. P. L. F. DE CARVALHO
Computing Department, University of São Paulo at São Carlos, São Carlos, SP CP 668, CEP 13560-970, Brazil

M. C. FAIRHURST, D. L. BISSET
Electronic Engineering Laboratories, University of Kent at Canterbury, Canterbury, Kent, CT2 7NT, England

GSN^f is a Boolean Neural Network designed to be applied for pattern recognition tasks. This chapter presents the learning algorithms which have been proposed to train GSN^f architectures and compare their performances as key parameters are changed. These algorithms are evaluated against each other by taking into account training time, saturation, learning conflicts and correct recognition rates.

1 Introduction

The classification performance achieved by a neural network is a direct consequence of the correct development of a function that efficiently implements the mapping between input patterns and their associated desired outputs. This function development is very much dependent on the learning algorithm used to train the network. An inadequate teaching strategy makes this achievement very difficult, if not impossible. The choice of the appropriate strategy may depend on the learning requirement of the classifier. These requirements can be, for example, speed, learning capability and/or recognition performance.

Learning algorithms for Boolean neural networks work basically by changing the network connections [1,2], the Boolean function performed by the nodes [3] or the values stored in the memory contents [4,5]. The learning algorithms used by GSN^f are concerned with when and where the node's desired output (0 or 1) must be stored in the location(s) addressed by its input terminals.

This chapter investigates learning strategies available for GSN^f neural networks. For such networks, an extensive analysis is presented where the relative merits of various possible algorithms are evaluated. The algorithms evaluated are the Conventional algorithm [6], the Deterministic algorithm [7], a lazy approach to these algorithms [8] and a progressive algorithm [5]. These algorithms will be referred to as C, D, C^{lazy}, D^{lazy} and P respectively. This investigation will compare the performance achieved by these learning strategies in the recognition of machine printed characters extracted from mail envelopes. This evaluation will consider different aspects to measure the performance: learning speed, recognition rates and memory saturation rates.

2 GSN^J networks

GSN^J is a multi-layer architecture where GSN nodes are typically organised into several independent pyramids and the processing is performed by such a system in essentially three different phases. These are a validating phase, where the node determines which values it is capable of learning, a learning phase, where defined values can be stored in memory locations holding undefined values, and a recall phase, where the node generates as output the predominant value stored in the memory contents addressed from the input.

GSN nodes can accept, store and generate values of 0, 1 and u, where u represents an "undefined" value. If there is at least one undefined value on the input terminals of a GSN unit then a set of memory contents, rather than a single storage location, will be addressed, and this is a principal factor in the efficient distribution of stored information across a trained network.

The processing of GSN^J can be divided into three phases, each associated with a specific goal: a validating phase, a learning phase and a recall phase. The purpose of the validating phase is to produce a validating value for each pyramid. A validating value is the pyramid's output guess for a given input pattern. A pyramid's validating value defines which defined value(s) it can learn in the next learning phase, and a pyramid cannot be taught when its validating value is the opposite to its desired output value. The learning phase teaches the pyramids by changing the values stored in the memory locations of their constituent nodes. The recall phase produces the pyramids' output for an unknown input, seeking, for each node, to output the defined value with the greatest occurrence frequency in the addressable memory contents. The following subsections will explain in more detail each of the three states.

2.1 Validating state

The goal in this state is to determine the values that the node can learn in the next learn phase without disrupting previously stored information. In order to discover this, the input to be learnt is fed into the input terminals. A function is applied to the values stored in the GSN addressed memory contents to determine its validating value. This validating value is propagated through the pyramid to its apex. The validating value produced by the node on the apex will be the pyramid validating value. If the output is equal to u (undefined) then it can learn any desired output. If, on the other hand, the output is a defined value (0 or 1), only this value can be taught. The output of a node n_i in the validating state is given by the Equation 1 below:

$$o_i = \begin{cases} 0 & \text{iff } \forall a_{i_m} \in A_i, C_i[a_{i_m}] = 0; \\ 1 & \text{iff } \forall a_{i_m} \in A_i, C_i[a_{i_m}] = 1; \\ u & \text{iff } \exists a_{i_m} \in A_i \mid C_i[a_{i_m}] = u \text{ or } \exists a_{i_m}, a_{i_l} \in A_i \mid C_i[a_{i_m}] \neq C_i[a_{i_l}]. \end{cases} \quad (1)$$

In this equation, A_i represents the set of addresses originated from the input, $C_i[a_{i_m}]$ and $C_i[a_{i_l}]$ are the contents of n_i that are accessed by addresses a_{i_m} and a_{i_l} respectively.

According to Equation 1, the output generated in the validating state can be equal to 0, 1 or u. When all the values stored in the addressable contents are equal to 0, the output will be equal to 0. If, on the other hand, all the values stored in the addressable contents are equal to 1, the output will be equal to 1. The output will be equal to u in any other case.

2.2 Learning state

The goal in the learning phase is to store the desired output in a location addressed by its input terminals. To use its cells more efficiently, GSN tries to store its desired output in an addressed memory content which already stores this value. When there is more than one option, a memory location holding an undefined value is selected. Equation 2 illustrates the choice of the content.

$$a_{i_m} = \begin{cases} Ran(A_{i/_{d_i}}) & \text{iff } \|A_{i/_{d_i}}\| > 0; \\ Ran(A_{i/_u}) & \text{iff } \|A_{i/_{d_i}}\| = 0. \end{cases} \quad (2)$$

where $A_{i/_s} = \{a_{i_k} \in A_i \mid C_i[a_{i_k}] = s\}$ represents the addressable set of the node n_i that stores the value s, $Ran(A_{i/_s})$ is a element randomly chosen from $A_{i/_s}$ and $\|A_{i/_s}\|$ is the number of elements that belong to $A_{i/_s}$.

As can be seen in Equation 2, a GSN node searches for a cell in its addressable set which stores a value equal to its desired output. If there is no such cell, it chooses a cell storing an undefined value. If, for either of the two cases mentioned above, there is more than one possible choice, a cell from the addressed set is chosen at random. After choosing the cell, the address of the cell selected must then be sent back, through the desired input terminals, to be used as desired output by the nodes in the previous layer. The desired input d_{ij} that must be sent to the node n_j in the previous layer is the j^{th} bit

of the address a_{i_m}. Thus each node n_j receives its desired output, learns and provides desired outputs to nodes in the previous layer.

2.3 Recall state

When the node gets to its recall state, it has as the goal to produce the value with the highest occurrence in the addressable set. This value is propagated through the pyramid layers in a similar way as the validating value to produce the pyramid recall value. The method used to define the recall value is given by Equation 3.

$$o_i = \begin{cases} 0 & \text{iff } \|A_{i/_0}\| > \|A_{i/_1}\|; \\ 1 & \text{iff } \|A_{i/_1}\| > \|A_{i/_0}\|; \\ u & \text{iff } \|A_{i/_0}\| = \|A_{i/_1}\|. \end{cases} \tag{3}$$

In this equation, the node will generate as output a value 1 only if the number of such values in the addressable locations is greater than the number of 0's. If the inverse occurs (i.e., the number of 0's is greater than the number of 1's), the output will be equal to 0. It does not matter how many undefined values exist in the addressable locations. When the addressed memory contents store the same number of defined values 0 and 1, the output will be an undefined value. This rule has the effect of minimizing the propagation of undefined values.

3 Learning algorithms

The C and D learning algorithms were the first strategies developed to teach GSN^f neural networks. Their only difference is the approach taken for selecting the memory content to store the desired output value when more than one option exists. While the C algorithm, chooses the memory position at random, the D algorithm, chooses a memory position whose address has a larger number of bits in common with the defined value to be learned. According to [7], the purpose of the D algorithm is to make the internal representations of patterns from the same class as similar as possible and vice-versa. As a consequence, the network increases the discrimination between patterns of different classes and reduces the discrimination between members of the same class.

By using these learning strategies, GSN^f has achieved very good recognition performance when applied to character recognition [6,5,7]. Its one-shot learning characteristic makes it particularly suitable for pattern recognition

applications requiring fast learning. However, an analysis of the learning algorithms employed by GSN has shown that they teach the network more than is necessary. This happens in two different ways. First, memory contents can be taught with defined values that are already stored there, i.e., the previous learning algorithms store a defined value in a memory content of every node in the pyramid they are teaching, even if these memory contents already store the same defined value. Second, memory contents holding undefined values are unecessarily updated when they could be kept unchanged for future need. Memory contents holding undefined values are unncessarily updated when they could be kept unchanged for future need.

3.1 Lazy learning

The lazy learning strategy overcome this problem by reducing the storage of defined values in the memory contents to a minimum, decreasing saturation, making learning conflicts less probable and speeding up the learning process. These improvements are achieved by avoiding teaching a node each time its desired output is equal to its validating value or the node to which its output is connected was not taught. Thus, if a node on the apex of a given pyramid is not taught, the whole pyramid is not taught.

In fact, this kind of teaching is unnecessary because if the validating value v_{ij} generated by a given node n_{ij} in a pyramid P_i is a defined value, then all the nodes n_{ik} in the sub-pyramid p_{ij} (with the node n_{ij} on the apex) whose validating value is undefined can be taught with any defined value. The defined value chosen does not cause any change to the value that would be generated for the same input pattern in the recall phase, since if a defined value is generated for a pattern in the validating phase, due to the nature of the recall and validating function[6], it will necessarily be generated by the same pattern in the recall phase.

Figure 1 shows which nodes of a particular pyramid are taught when learning algorithm D is used with and without laziness. According to this figure, the three sub-pyramids which produced a defined validating value do not need to learn. This lazy learning approach has been used with the two previous learning algorithms (C and D) to generate two new learning algorithms to train GSN^f architectures (C^{lazy} and D^{lazy}).

3.2 Progressive learning

The previous learning algorithms compare the validating value to the desired output value and then, according to this comparison, train each pyramid in a backward direction, starting at the apex node and working back to the pyramid

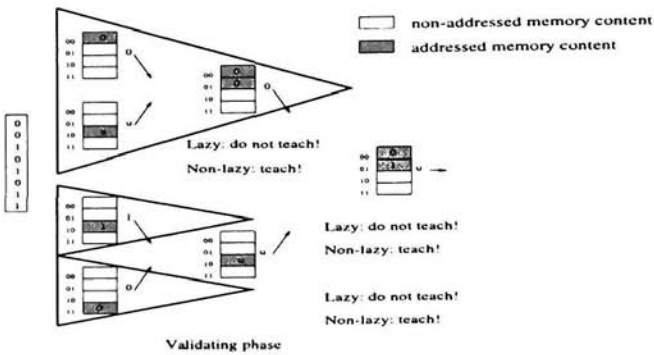


Figure 1: The effect of laziness in the nodes teaching

base. This characteristic makes the implementation of GSN^f more complex and limits the node fan-out to one. These problems are overcome by the Progressive learning algorithm.

The Progressive algorithm does not use the validating phase and runs in the opposite direction, starting the training with the nodes in the base and, progressively, teaching all the nodes in each layer up to the apex. During the training of the first layer of a given pyramid P_i, all the memory positions of this layer which are addressed by the input pattern and store undefined values have their value changed to the desired output value associated with P_i. At successive layers, outputs of the previous layer produce the inputs to the next. Besides, the P algorithm makes it possible to change defined values stored in the memory content without a large loss of information, what was not the case with the other algorithms. Another advantage of the P algorithm is that the nodes with a fan-out larger than 1, and structures other than pyramids, can be used. These different learning strategies are evaluated in the following section.

4 Experiments

Experiments have been carried out to evaluate the influence of each of these algorithms on the performance of GSN^f considering a number of benchmarks found to be of importance in practical systems. The parameters of comparison adopted in this comparative study are the saturation rate of each pyramid layer, the training time and the recognition performance. For such, GSN^f networks comprising 100 3-layer pyramids have been used. In order to make the results achieved more representative, they are shown for connectivities 2, 4 and 6. A machine printed data set with numerals extracted from mail

Table 1: Classification accuracy for different connectivities

GSN^J Classification accuracy (%)					
Connectivity	*Learning strategy*				
	C	D	C^{lazy}	D^{lazy}	P
2	93.80	92.76	94.55	92.91	93.93
4	84.07	93.04	84.93	93.61	93.53
6	66.94	95.92	68.52	95.85	96.36

envelopes has been used. The networks were trained with 10 samples from each numeral and tested with 100 additional sample of each numeral. All the results represent the average of 10 similar experiments run with different input mappings.

4.1 Recognition performance

The benchmark usually adopted when a pattern recognition method is being evaluated is its recognition performance. The recognition performance measures how efficiently the method discriminates between patterns from different classes and how it generalizes when presented with patterns from the same class. Table 1 presents the recognition rates achieved for each learning algorithm.

It is clear from Table 1 that the increase of the connectivity has different effects in the recognition performance for the five strategies. While in the conventional-based algorithms the increase of the connectivity leads to a decrease in performance, the other algorithms present a clear improvement in performance when larger connectivities are used. It should also be noted that the progressive algorithm is the one which provides the highest performance of all the cases when connectivity 6 is used. Progressive learning provides better performance because it increases the pyramids's generalization ability by updating all addressed memory contents holding undefined values of each pyramid with the same desired output value for the patterns of the same class.

4.2 Saturation

One of the main problems faced by Boolean neural architectures is the saturation of their nodes's memory contents during the learning phase. This problem is dependent on the relation between the storage capacity and the number of patterns per class or the number of classes. The teaching of each new pattern

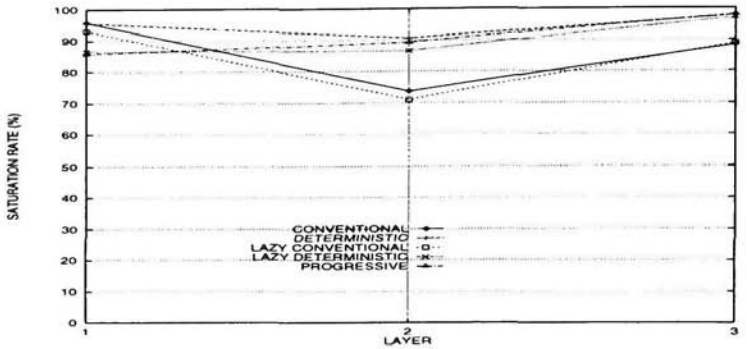


Figure 2: Saturation rates for connectivity 2

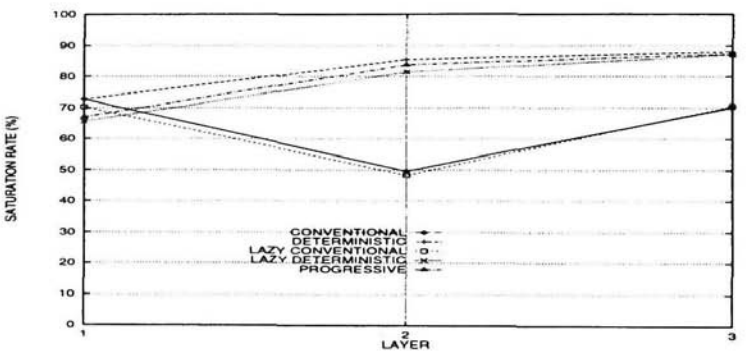


Figure 3: Saturation rates for connectivity 4

reduces the number of memory contents available and this is even more critical when low connectivity is used, since a small number of memory contents are accessible. If not enough memory contents are available, there will be a point where just a few or even no more new patterns can be learned. When one-shot learning is used, this problem is even worse because the earlier patterns will be better represented than the later ones. Figure 2, Figure 3 and Figure 4 show the saturation rates for each pyramid layer when connectivities 2, 4 and 6 are used.

With connectivity 2, it can be seen that, for the C and C^{lazy} cases, the first and the last layers have a much higher saturation than the intermediate layer. With D, D^{lazy} and P, the three layers present more similar saturation rates. The third layer yields a higher level of saturation than the second layer

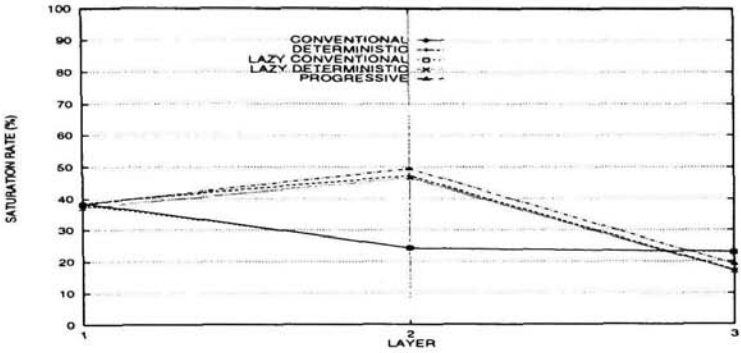


Figure 4: Saturation rates for connectivity 6

for all the cases. According to[9], the ideal situation is one where the saturation rates are evenly distributed among all the layers. When connectivity 4 is used, all five algorithms present a lower saturation rate than in the previous case, which is due to the larger number of more memory contents available. However, while the saturation rates for the C and C^{lazy} algorithms keep a similar relative proportions in all the three layers for different connectivities, when the D, D^{lazy} and P are used, the saturation rates for the second layer become relatively higher than in the first layer.

The saturation rates for the second layer become higher than in the first because of the larger increase in the number of memory contents in the first layer when the overall connectivity is increased. This unbalanced increase in the number of memory contents affects the C and C^{lazy} algorithms in a smaller scale. By using connectivity 6, the saturation rates of all layers, in particular the last layer, are decreased. This decreased is expected because of the higher number of memory contents available.

4.3 Learning conflicts

The concept of conflict during the learning process is a problem that can be faced by a GSN^J architecture every time a new pattern has to be learned. A conflict of learning occurs when the validating value produced by the validating phase for a given pyramid P_i is the opposite of its desired output value. Conflicts are due to:

- Impossible mappings because of limitations in the functionality of pyramid structures.

Table 2: Conflict rates for different connectivities

GSN^J learning conflicts (%)					
Connectivity	*Learning strategy*				
	C	D	C^{lazy}	D^{lazy}	P
2	23.63	19.56	23.14	19.31	18.11
4	13.83	7.90	13.42	6.91	7.12
6	0.63	0.00	0.65	0.00	0.01

- The same pattern in the area covered by P_i being associated with different desired outputs.

There are, however, various possible ways to overcome, or at least reduce, these conflicts during the learning process. The first cause can be reduced or eliminated either by increasing the number of layers maintaining the same covered area, by covering some pixels more than once, using nondisjoint pyramids [3], or by increasing the node's connectivity. The second problem can be handled by increasing the area covered by each pyramid. A combination of these two techniques could reduce or solve both of the problems. Another alternative to solve both problems is to redefine the desired output for the pyramid in order that similar patterns would have the same desired output, regardless of which classes they belong to.

The number of conflicts is largely affected by the saturation rates. If the saturation rates are higher, fewer memory contents are available to be updated by the learning algorithm and more learning conflicts are likely to occur. Nonetheless, higher saturation rates may indicate that more patterns have been learned, and thus less learning conflicts have occurred. Table 2 illustrates the percentage of learning conflicts for each learning strategy when connectivities 2, 4 and 6 are used.

Looking at the results achieved with each connectivity, it is shown that although the D, D^{lazy} and P algorithms produce higher saturation rates than the C and C^{lazy} algorithms, they have a smaller number of conflicts. Making a better use of the memory contents available, the first three algorithms can teach more patterns by increasing the similarity of the internal representation of patterns with the same desired output in each pyramid, which reduces the number of conflicts.

Comparing the number of conflicts for the D and D^{lazy} cases and for the C and C^{lazy} cases, it can also be seen that the use of laziness slightly reduces the occurrence of learning conflicts. This is to be expected since, the lazy

Table 3: Training times

GSN^f *Training time (time units)*					
Connectivity	*Learning strategy*				
	C	D	C^{lazy}	D^{lazy}	P
2	5.70	8.00	3.00	3.10	3.00
4	34.00	53.10	22.00	26.00	21.00
6	206.10	323.50	156.30	219.20	139.50

strategies save memory contents for future patterns, allowing more patterns to be learned. As the connectivity increases, the percentage of conflicts produced by each learning strategy clearly decreases. Larger connectivity means more memory contents available, and therefore lower probability of not having a memory content available when one is needed.

4.4 Training time

Fast learning algorithms can be a crucial aspect in some real time applications. The one-shot learning approach used by GSN^f learning strategies leads to very small training times. Table 3 shows how GSN^f learning algorithms affect the time the network takes to be trained.

It can be seen from Table 3 that for every increase in the connectivity there is a much larger increase in the training time, which should be the case, because the number of memory contents of a node is the square of the nodes connectivity. But even so, the increase in the training time is much higher than the increase in the number of memory contents because additional operations are carried out and more undefined values are propagated during the validating phase. These results also demonstrate that the C^{lazy} and D^{lazy} algorithms are much faster than their original C and D algorithms and that the both the D-based approaches are slower than their correspondent C-based approaches. Finally, it shows that for any connectivity, the smallest training time is obtained by using the P approach.

5 Conclusion

In this chapter five learning algorithms available to train GSN^f neural networks are described and investigated. The algorithms have been investigated with respect to a number of important criteria: recognition performance, memory saturation and training time. It has been found that, usually, the larger the

connectivity, the better the recognition performance. The exceptions are the cases where the C and C^{lazy} algorithms have been used. They also show that the increase in connectivity decreases the saturation but increases the training time. The results achieved show that the lower saturation rates are obtained by the C and C^{lazy} cases, the shortest training time by the P and C^{lazy} cases and the best recognition performances by the D^{lazy} and P cases.

A. de Carvalho would like to acknowledge the support of the Brazilian Research Agencies CNPq and FAPESP

References

1. J. R. Doyle. Supervised learning in n-tuple neural networks. *Int. J. Man-Machine Studies*, 33:21–40, 1990.
2. S. Patarnello and P. Carnevali. Learning networks of neurons with Boolean logic. *Europhysics Letters*, 4(4):503–508, 1987.
3. W. W. Armstrong and J. Gecsei. Adaptation algorithms for binary tree networks. *IEEE Transactions on Systems, Man, and Cybernetics*, 9:276–285, 1979.
4. W. K. Kan and I. Aleksander. A probabilistic logic neuron network for associative learning. In *Proceedings of the IEEE First International Conference on Neural Networks*, volume II, pages 541–548, San Diego, California, June 1987.
5. A. de Carvalho, M. C. Fairhurst, and D. L. Bisset. Progressive learning algorithm for GSN feedforward neural architectures. *Electronics Letters*, 30(6):506–507, March 1994.
6. E. Filho, M. C. Fairhurst, and D. L. Bisset. Adaptive pattern recognition using goal-seeking neurons. *Pattern Recognition Letters*, 12:131–138, March 1991.
7. R. G. Bowmaker and G. G. Coghill. Improved recognition capabilities for goal seeking neuron. *Electronics Letters*, 28:220–221, 1992.
8. A. de Carvalho, M. C. Fairhurst, and D. L. Bisset. A lazy learning approach to the training of GSN neural networks. In *Proceedings of the ICANN 92, Elsevier*, pages 673–676, Brighton, UK, September 1992.
9. E. Filho, M. C. Fairhurst, and D. L. Bisset. Analysis of saturation problem in ram-based neural networks. *Electronics Letters*, 28(4):345–347, February 1992.

THE ADVANCED UNCERTAIN REASONING ARCHITECTURE, AURA†

J AUSTIN, J KENNEDY, K LEES
Advanced Computer Architecture Group,
Department of Computer Science, University of York, York, YO1 5DD, UK

The ADAM binary neural network which has been used for image analysis applications, is constructed around a central component termed a Correlation Matrix Memory (CMM). A recent re-examination of the CMM has led to development of the Advanced Uncertain Reasoning Architecture (AURA). AURA inherits many useful characteristics from ADAM, but is intended for applications requiring the manipulation of symbolic knowledge. This chapter shows how the AURA architecture has been developed from ADAM and explains its method of operation. The chapter also outlines the use of AURA in symbolic processing applications, and highlights some of the ways in which the AURA approach is superior to other methods.

1 Introduction

The ADAM neural network[1] is a binary network used for image analysis applications which is constructed around a central component known as a Correlation Matrix Memory (CMM) network. Our recent work has been examining the CMM networks in ADAM and has developed a more thorough understanding of the way the network operates. Through this, we have been able to develop the Advanced Uncertain Reasoning Architecture (AURA), which is intended for use in knowledge based systems tasks[3]. AURA is similar to ADAM in its ability to deal with uncertain data and simple implementation in hardware. AURA is also expected to share the ability of ADAM to operate at speed on very large amounts of data. In Section 2 we review the basic ADAM architecture and examine the central role played by the Correlation Matrix Memory networks in ADAM. In Section 3 we describe the structure, operation, and concepts of the AURA architecture. Concluding comments appear in Section 4.

2 Background

ADAM is an associative neural network intended primarily for image processing problems. It uses binary or grey scale N tuple methods[2] to allow large images and complex objects to be recognized. The development of the ADAM architecture was

† This work is supported by EPSRC and British Aerospace under grant number GR/K 41090, Advanced Architectures to Support Intelligent Command and Control.

intended to overcome limitations in more basic CMM structures. In particular, simple CMM networks tend to suffer from poor storage capacity due to an upper bound imposed by the size of input and output patterns, and because of interference between stored patterns (analogous to cross-talk).

2.1 *Outline of the ADAM Architecture*

The basic ADAM architecture is shown in Figure 1 and comprises a number of distinct stages: an N-tuple decoder, a first stage CMM network, a class separator, and finally a second stage CMM network. Use of two CMM networks allows the storage capacity of the system to be determined independently of the input and output pattern sizes. Details of the ADAM architecture can be found in[1].

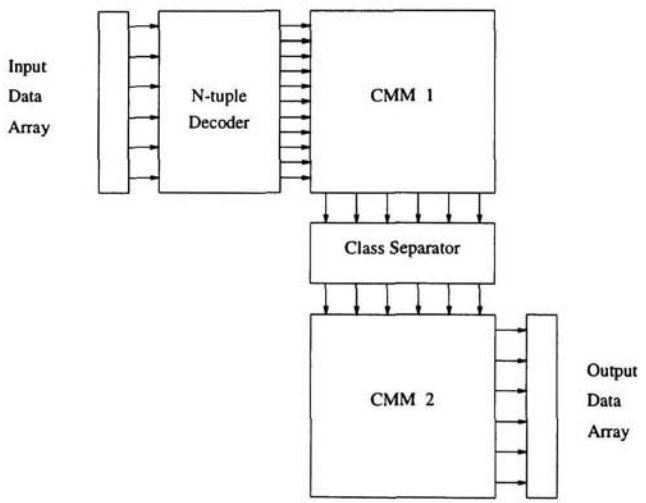


Figure 1: The ADAM Architecture

2.2 *Correlation Matrix Memory Networks*

Our recent work has focused on the way the CMM networks operate within ADAM. The binary CMM network is a key functional element of the ADAM system which provides the networks associative capability. A CMM network can be regarded as a single-layer weightless neural network, but is most easily seen in terms of a binary matrix W. Thus each element w_{ij} of the matrix W takes the value 0 or 1 to represent the absence or presence (respectively) of an unweighted synaptic connection between corresponding input and output units.

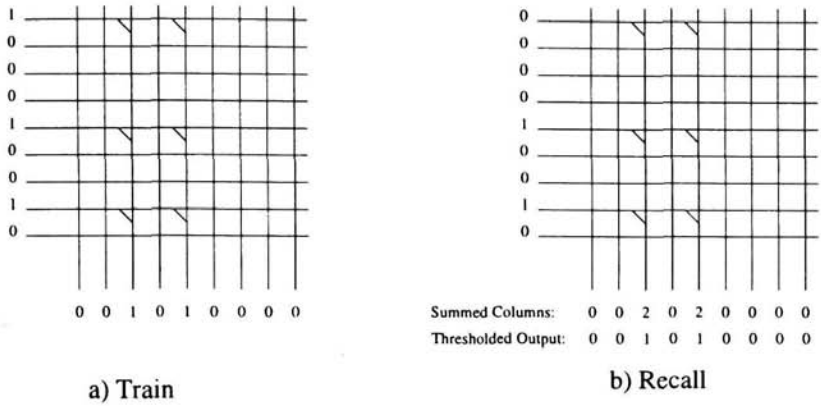


Figure 2: A Correlation Matrix Memory Network

Figure 2 shows an example of a small CMM network in both train and test modes. In Figure 2 a training is accomplished by creating an association between two binary patterns using a simple Hebbian learning procedure. The two patterns are applied to the rows and columns of the CMM network simultaneously. An element of the matrix is set to 1 corresponding with the intersection of active rows and columns. (A row or column is active when a bit is set in the applied pattern at that position.) In mathematical terms this is equivalent to computing the outer product of the two patterns to obtain the matrix W (i.e. $W = uv^T$, where u is the (input) column vector and v^T is the (transposed, output) row vector).

In Figure 2b a pattern is applied to the rows of the CMM network trained in Figure 2a. This pattern is similar to the pattern used in training except that the bit applied to the first row is no longer set (this could have occurred due to noise, for example). Recall is accomplished by summing the active row connections in each vertical column. An appropriate thresholding function then recovers the original binary pattern, first applied to the columns in Figure 2a. Thus a CMM network can be made robust against corrupted input patterns during recall by a suitable choice of thresholding function (both *L-max* and *Willshaw* thresholding are used in AURA[3]). This property is exploited in AURA to provide a powerful partial matching mechanism for incomplete inputs (see Section 3.6).

It is clear that the input to a CMM network need not be N tuple states as in ADAM, but any pre-processed data. CMM networks have been used to represent frame-based knowledge, and to implement multiple-hypothesis generation, similarity matching, and hypothesis selection in an experimental system based on the original ADAM architecture[7]. Building on these ideas, we have developed methods for pre-processing

and presenting rule based data[3] which have led to the new AURA architecture.

3 The AURA architecture

The AURA architecture can be used to form the basis of high performance knowledge based systems. AURA provides facilities to enable high-speed matching between environment variables and a knowledge base consisting of rules that typically have the form: *antecedent-expression* → *consequent-expression*.

The architecture exploits particular properties of CMM networks to support evaluation of which generally contains bound symbolic references to environment variables. In addition, a powerful form of partial matching is performed with incomplete expressions (in which some environment variables are not available). However, in this case *consequent-expression* should be treated as a tentative, uncertain conclusion, to be confirmed by subsequent processing.

The system is first trained with a knowledge-base, i.e. a set of rules or predicates. These specify particular pre-conditions and (via class separator patterns) the consequential actions to be taken if the pre-conditions are satisfied. In operation, values become bound to environment variables, forming the input to AURA. Thus rules in the knowledge-base become eligible to be "fired" when the configuration of variables satisfies the pre-conditions of those rules.

3.1 AURA Structure

The principal components of the AURA architecture are shown in Figure 3. The architecture is based on the use of an array of CMM weightless neural networks, supported by mechanisms which: a) convert lexical tokens into binary pattern vectors with exactly k bits set (where k is a constant for a given CMM network), b) perform binding of variable-names to values, c) form superimposed codings of sets of bound variables, d) route the superimposed sets to appropriate CMM networks, and e) resolve multiple network outputs (which occur in the form of superimposed class separator patterns). The *lexical converter* replaces the N tuple pre-processing used in ADAM, converting symbolic and numeric data to fixed weight binary patterns. This is, in effect, what N tuple pre-processing does to image data.

3.2 Rule Representation

The current version of AURA is intended to process rules expressed in a general boolean sum-of-products form such as: $A.B + B.C + D \rightarrow S$. Each conjunctive expression such as $A.B$ is treated separately in AURA as though there were in fact three rules, each containing only boolean product terms: $A.B \rightarrow S$, $B.C \rightarrow S$, and $D \rightarrow S$. There are particular architectural advantages following form the use of this equivalent rule form.

This leads to considerable simplification in the representation and matching of complex rule conditions.

Another simplification in the current architecture is that the consequent or "action" part of a rule is not represented explicitly in the CMM networks. Instead, this is represented by a single (system generated) symbolic pattern referred to as a *class separator* pattern (or just *separator*). In general, multiple separators are produced by the rule matching process (due to multiple rule matches). These appear as superimposed binary patterns, similar in form to those input to the CMM network (see Section 3.4). To recover the actual consequent parts of rules, it is first necessary to identify all valid separators occurring in the multiple output. Although a further CMM network can be used for this purpose there are simpler, more rapid methods for performing this specialised task (for example, the *middle-bit indexing* method proposed in [9]).

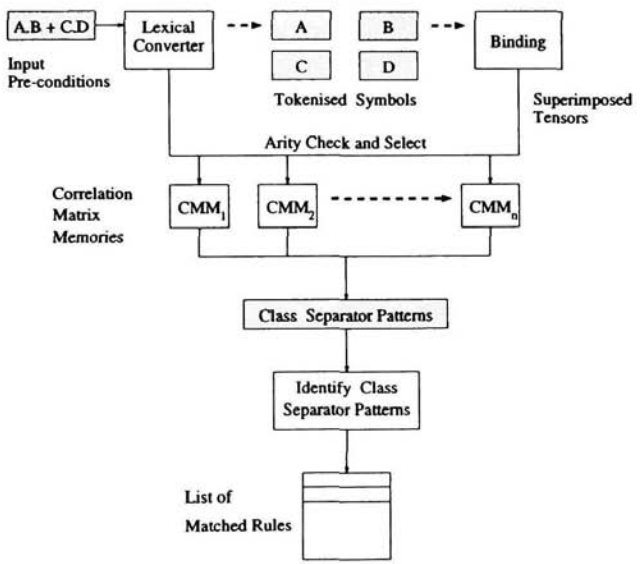


Figure 3: The AURA Architecture

3.3 Variable Binding

In symbolic processing it is necessary to bind symbolic variable names to the values they represent. This permits evaluation of general symbolic expressions (e.g. a rule pre-condition) when instantiated with particular bound values. In the past, it has often been found difficult to implement symbol bindings in neural networks. Space does not

permit a description of the discussion surrounding this binding problem here, but the approach taken in AURA appears to overcome some of the problems encountered previously. The approach taken is similar to[4] but uses binary instead of real-valued tensors.

In the AURA architecture, tensor products are used to bind variables to values. Tensor product (TP) vectors can be formed in two steps for a pair of binary patterns representing the symbolic name and the value to which it is bound. The first step involves calculation of a TP matrix and is equivalent (in the two pattern case) to the outer product computation used to store a pair of patterns in a CMM network. In the second step, the TP vector is obtained by simply concatenating the rows of the matrix obtained in the first step.

3.4 Commutativity of Bound Symbols

In principle, we could simply concatenate the TP vector patterns representing bound variables in rule pre-conditions. This leads to difficulties in practice, because the size of a given CMM network would become dependent on the complexity of the rules with which it had been trained. Perhaps a more serious limitation with concatenation is that symbols in rule condition expressions are no longer commutative. Thus each binary pattern in an expression would need to be routed to the correct position in a concatenated input field representation.

An alternative approach adopted in AURA is to use a superimposed coding of the binary patterns representing bound variables in rule condition expressions. A similar technique has been popularly used in database systems in forming compact keys[6]. The approach consists simply of superimposing the binary patterns by computing a bitwise OR over all patterns in an expression.

3.5 The CMM Array

The primary functional element in the AURA architecture is a rule-matching engine implemented as an array of CMM networks. The need for multiple CMM networks arises because of difficulties in defining an appropriate thresholding criterion when rules of different *arity* are stored in the same CMM network. Here, rule *arity* refers to the number of boolean product terms occurring in the conjunctive expression of the rule condition (as described in Section 3.2).

For example, suppose that two rules $A \rightarrow S_1$ and $A.B \rightarrow S_2$ are stored in a single CMM network. Later, if only the bound variable A is applied to the CMM network (if say, the value of B is not yet known) we find that the separator patterns for both rules receive the same degree of activation, even though the rule $A.B \rightarrow S_2$ is only matched partially for this input.

This problem can be avoided if the rules are stored in a different CMM network

determined according to rule arity. The use of multiple networks allows full control over the matching process: a unique CMM network is allocated for the storage of rules of each required arity. Thus a rule such as $A.B.C \rightarrow S$ which has an arity of three, would be stored in the CMM allocated for rules of arity three.

3.6 Partial Matching

A major aim in the design of the AURA architecture has been the provision of a special form of partial match, not generally found in other systems. The usual form of partial or incomplete match which is provided by many systems (including AURA) involves matching *one particular* combination of terms (n) from the full set of terms or attributes (m) in a previously stored rule (or record).

The AURA architecture extends this idea, providing a second, more general version of the partial match which provides much greater flexibility in the matching process. In this version, we can specify that *any combination* of n terms from m is permitted in a successful match. For example, we can choose to accept a rule of arity five which matches on any three of the five boolean product terms in the rule. To perform this second type of match, most systems would require ${}_mC_n$ separate match operations, whereas in AURA this is performed in a single match operation.

Since partial matching is achieved by manipulating the threshold in AURA, it is simple to dynamically relax the criteria for a successful match, in effect by reducing the specified value of n. This combines the two versions of partial match described above, and can be useful (for example) when the initial value of n fails to produce a successful match.

3.7 Applications

The AURA architecture is primarily aimed towards applications requiring rapid manipulation of large knowledge bases. A subsidiary aim is to provide a degree of support for uncertainty management in such applications by exploiting the partial match capability of the architecture.

An important source of motivation for this work is the requirements of mission management systems as found, for example, in future generations of aircraft. Such systems constantly demand performance improvements in command and control functions including data fusion, situation assessment, and sensor management. In most cases, the performance of these functions can be considerably enhanced with support from high-speed rule evaluation. For example, in the situation assessment function, generic situations could be recognised rapidly using rules. The special partial match abilities of AURA could prove extremely useful in "filtering out" small variations between similar situations.

4 Conclusions

The main objective of the work described here is to develop novel architectures for knowledge manipulation. One of the most important aims of the work is to develop systems that can be implemented efficiently in hardware. Towards this end, we have developed an architectural framework based on weightless correlation matrix memory neural networks. A set of dedicated support chips is under development[5] to enable the construction of high-performance systems based on AURA concepts.

The major benefits of the architecture described here are: 1) efficient rule matching in knowledge bases, 2) powerful partial match mechanism, 3) simple hardware implementation. The system is being applied to rapid reasoning in aircraft systems, where it is necessary to support all these features.

References

1. J Austin, "ADAM: A Distributed Associative Memory For Scene Analysis," Proceedings of the First International Conference on Neural Networks, Eds. M. Caudill and C. Butler, Vol. IV, pp. 285-292, June 1987.
2. J Austin, "Distributed Associative Memories for High Speed Symbolic Reasoning," International Journal of Fuzzy Sets and Systems, Ed. N Kasabov, (accepted for publication).
3. WW Bledsoe and I Browning, "Pattern Recognition and Reading by Machine," Proceedings of the Eastern Joint Computer Conference, pp. 225-232, 1959.
4. R Filer, "Symbolic reasoning in an associative neural network," Master's Thesis, Department of Computer Science, University of York, September 1994.
5. T Jackson, J Austin, "The Representation of Knowledge and Rules in Hierarchical Neural Networks," In: *Neural Networks for Knowledge Representation and Inference* (Ed. DS Levine and M Aparicio IV), Lawrence Erlbaum Associates, Hillsdale NJ, 1994.
6. J Kennedy and J Austin, "A hardware implementation of a Binary Neural Network,", Fourth Int. Conf. on Microelectronics for Neural Networks and Fuzzy Systems, IEEE Computer Press, pp. 178-185, 1994.
7. R Sacks-Davis, K Ramamohanarao, "A Two Level Superimposed Coding Scheme for Partial Match Retrieval," Information Systems 8(4), 273-280.
8. P Smolensky, "Tensor Product Variable Binding and The Representation of Symbolic Structure in Connectionist Systems," Artificial Intelligence 46, 159-216.

Section 2

Extensions to N tuple Theory

The following set of chapters describe extensions to the methods used in RAM based systems. As with all neural network methods, there is a continual aim to improve the performance of methods and undertake comparisons with other techniques.

The first five chapters describe new methods for the analysis of N tuple systems that allow the networks to be used more effectively. The first chapter by Morciniec and Rohwer presents a thorough comparison of the RAM based methods and other neural networks which clearly demonstrates that RAM based networks are at least as good as a wide range of other networks and statistical methods on a range of complex and well known benchmark problems. The next chapter shows that RAM based networks, although commonly thought of as binary networks are capable of using continuous inputs in the domain of image processing. The chapter by Howells, Bisset and Fairhirst describes, in general terms, how RAM based networks that use the GSN learning methods may be compared with and integrated with other RAM based methods. Jorgensen, Christensen and Liisberg show how the well known cross validation methods and information techniques can be used to reduce the size of the RAM networks and in the process improving the accuracy of the networks. Finally a very valuable insight into calculation of the storage capacity of a wide section of RAM based networks is given by Adeodato and Taylor. The general solution permits the capacity of G-RAM, pRAM and GSN networks to be estimated.

The final three chapters in section 2 describe new RAM methods which extend the basic ability of the networks. The chapter by Morciniec and Rohwer shows how to deal with zero weighted locations in weighted forms of RAM based networks. Normally this these are dealt with in an ad-hoc fashion. Although a principled approach is presented (based on the Good-Turing density estimation method), it is shown that using very small default values is a good method. It also contrasts binary and weighted RAM based approaches. The next chapter by Neville shows how a version of the Back propagation algorithm can be used to train RAM networks, allowing the RAM methods to be closely related to weighted neural network systems, and showing how Back propagation methods can be accelerated using RAM based methods. The chapter by Jorgsnsen shows how the use of negative weights in the storage locations allows recognition success to be improved for handwritten text classification. Finally, Howells, Bisset, and Fairhurst explain how the BCN architecture can be improved by allowing each neuron to hold more information about patterns it is classifying (which results in the GCN architecture) and by the addition of a degree of confidence to be added (which results in the PCN architecture).

This page is intentionally left blank

BENCHMARKING N-TUPLE CLASSIFIER WITH STATLOG DATASETS

M. MORCINIEC[†], R. ROHWER[§],
Neural Computing Research Group,
Aston University, Birmingham, B4 7ET, UK

The n-tuple recognition method was tested on 11 large real-word data sets and its performance compared to 23 other classification algorithms. On 7 of these, the results show no systematic performance gap between the n-tuple method and the others. Evidence was found to support a possible explanation for why the n-tuple method yields poor results for certain datasets. Preliminary empirical results of a study of the confidence interval (the difference between the two highest scores) are also reported. These suggest a counter-intuitive correlation between the confidence interval distribution and the overall classification performance of the system.

1 Introduction

The n-tuple classification system is one of the oldest neural network pattern recognition methods[5], and there have been many reports of its successful application in various domains[4,7,10,11,12]. The major advantage of the training set is its lightning speed. Learning is accomplished by recording features of patterns in a random-access memory, which requires just one presentation of the training set to the system. Similarly, recognition of a pattern is achieved by checking memory contents at addresses given by the pattern.

It is prudent to suspect that relatively poor performance will accompany the speed and simplicity of the n-tuple algorithm. We therefore carried out a large-scale experiment[8] in which the n-tuple method was tested on 11 real-word datasets previously used by the European Community ESPRIT StatLog project[9] in a comparison of 23 other classification algorithms including the most popular neural network methods. The results, reviewed below, show the n-tuple method to be a strong performer, except in a few cases for which we can offer explanations.

Statistics were also recorded on the confidence intervals (the differences between the two highest scores). Preliminary results suggest that two types of distribution occur, and performance is correlated to the distribution type.

[†] Current address: Hewlett-Packard Labs, Filton Rd., Stoke Gifford, Bristol BS12 6QZ, UK

[§] Current address: Prediction Company, 320 Aztec St., Suite B Santa Fe, NM, 87510, USA,
email: rr@predict.com

2 Selection and pre-processing of StatLog data sets

The StatLog project was designed to carry out comparative testing and evaluation of classification algorithms on large scale applications. About 20 data sets were used to estimate the performance of 23 procedures. These are described in detail in [9]. This study used 11 large data sets, selected as described in [8]. A specific random division into training and test sets was supplied for each data set.

The attributes of the patterns in the StatLog data sets are mostly real numbers or integers. Therefore each attribute was rescaled into an integer interval, quantised, and converted into a bit string by the method of Kolcz and Allinson [2,3] based on CMAC and Gray coding techniques.

The prescription for encoding integer x is to concatenate K bit strings, the jth of which (counting from 1) is $\frac{x+j-1}{K}$, rounded down and expressed as a Gray code. The Gray code of an integer i can be obtained as the bitwise exclusive-or of i (expressed as an ordinary base 2 number) with $i/2$ (rounded down). This provides a representation in aK bits of the integers between 0 and $(2^a - 1)K$ inclusive, such that if integers x and y differ arithmetically by K or less, their codes differ by Hamming distance $|x - y|$, and if their arithmetic distance is K or more, their corresponding Hamming distance is at least K. The resulting bit strings are concatenated together, producing an input vector of length $L = aKA$, where A is the number of attributes.

3 Benchmarking results

The benchmark tests used preprocessor parameters $a = 5$ and $K = 8$, so each attribute was scaled to the interval $[0, 248]$ and coded in 40 bits. (Test set attributes falling outside this range under the scaling based on the training set were truncated.) The recogniser used 1000 n-tuples of size $n = 8$, with 1 bit of memory at each address. In the event of a 'tie', meaning that the highest-scoring class had the same score as one or more other classes, the class among these with the highest *a priori* probability was selected.

Results from the benchmarking exercise are shown together with the Statlog results for other algorithms in Figure 1. Table 1 is a key to the symbols representing the various algorithms in this figure. Classification performance is normalised to the probability of the most common class, which equals the success rate obtainable by the trivial algorithm of always guessing that class.

The results show no systematic bias against the n-tuple method, except on the 4 data sets which tended to be the most problematic for the other algorithms. The geometric properties of the n-tuple method provide insight

Table 1: Synopsis of Algorithms with symbols used in Figure 1.

RAMnets	
(●) n-tuple recogniser	
Discriminators	
(♣) 1-hidden-layer MLP.	(♠) Radial Basis Functions.
(♡) Cascade Correlation.	(⊕) SMART (Projection pursuit).
(⊗) Dipol92 (Pairwise linear discrim.).	(⊖) Logistic discriminant.
(⊘) Quadratic discriminant.	(⊙) Linear discriminant.
Methods related to density estimation	
(α) CASTLE (Prob. decision tree).	(β) k-NN (k nearest neighbours).
(γ) LVQ (Kohonen).	(δ) Kohonen Topo. map.
(ε) NaiveBayes (Indep. attributes).	(ζ) ALLOC80 (Kernel functions)
Decision trees	
(a) NewID (Decision Tree)	(b) AC^2 (Decision Tree)
(c) Cal5 (Decision Tree)	(d) CN2 (Decision Tree)
(e) C4.5 (Decision Tree)	(f) CART (Decision Tree)
(g) IndCART (CART variation)	(h) BayesTree (Decision Tree)
(i) ITrule (Decision Tree)	

into the problematic cases.

4 Counting Hypercubes

It is well established that the tuple distance between two patterns (the expected number of tuples on which they differ) decays exponentially with Hamming distance according to

$$\rho(H) \approx N\left(1 - e^{-\frac{n}{t}H}\right) \tag{1}$$

to a good approximation [1,6]. Therefore each training pattern defines an A-dimensional hypercube with edges of Hamming length roughly aK/n in each attribute, over which it can contribute to the score of a test pattern. The preprocessor gives Hamming distances linear in arithmetic differences up to Hamming distance K of a possible aK per attribute, corresponding to arithmetic difference K of a possible $(2^a - 1)K$. Demanding this linearity throughout the region of near tuple distance gives $K > Ka/n$, or

$$a < n. \tag{2}$$

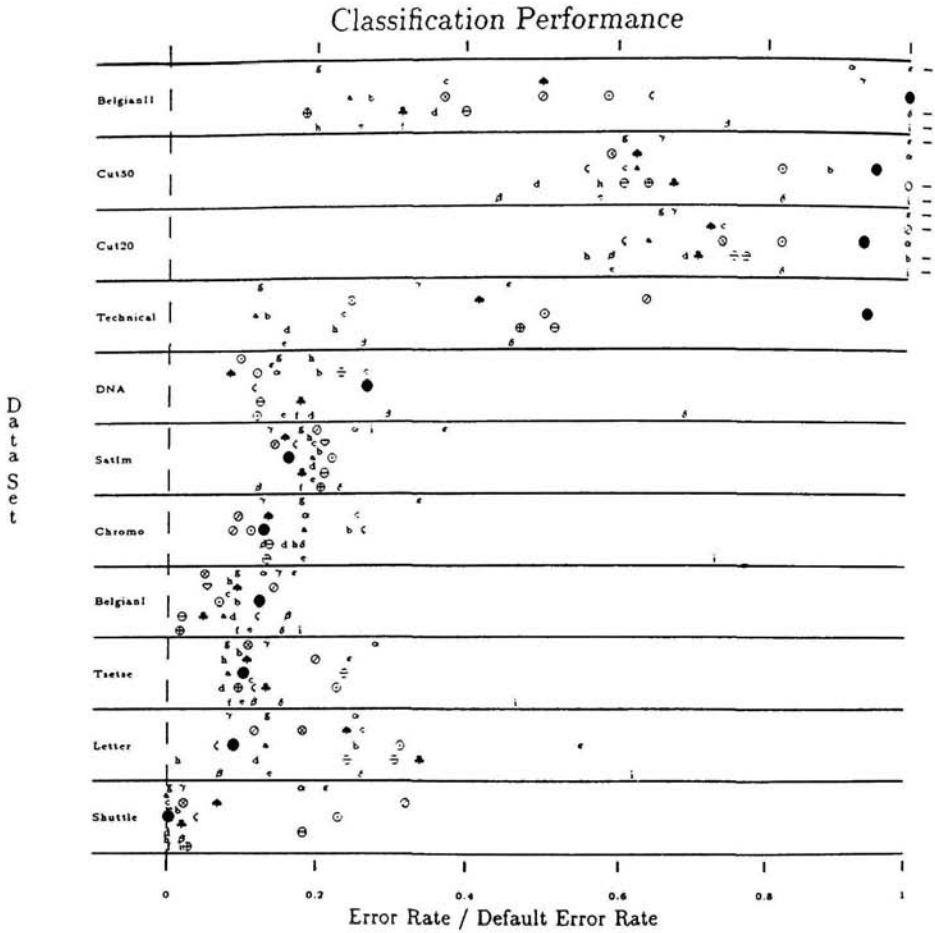


Figure 1: Results for N-tuple (●) and other algorithms. Classification error rates increase from left to right, and are scaled separately for each data set, so that they equal 1 at the error rate of the trivial method of always guessing the class with the highest prior probability, ignoring the input pattern. The arrows indicate performance which was worse than this.

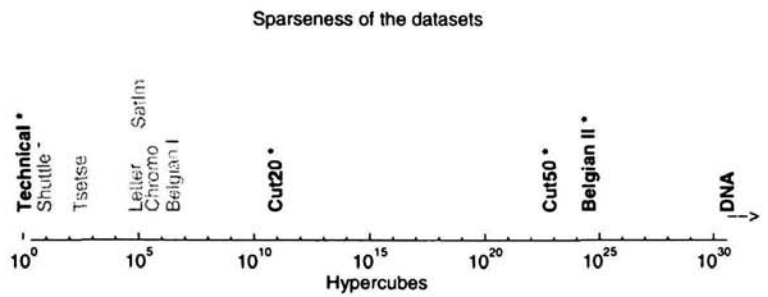


Figure 2: The number of hypercubes required to cover the space occupied by data. The datasets on which n-tuple classifier performed poorly are printed in bold face. A star denotes the existence of skewed priors. The DNA dataset had a highly redundant representation of its attributes and most of the data for Technical was concentrated in one hypercube.

The input region corresponding to a training pattern's hypercube consists of scalar differences up to $\pm K$ in each attribute, which is the fraction $\frac{2K}{(2^a-1)K}$ or about 2^{1-a} of the maximum separation $(2^a - 1)K$. Therefore each training pattern generalises to an input space hypercube whose volume is fraction $2^{-(a-1)A}$ of the total input space. Assuming a Gaussian distribution for the data, it is then possible to estimate the number of hypercubes need to cover the input region where data is likely to occur. The estimate is the product of the eigenvalues of the training data covariance matrix, in units of hypercube edge length, rounded up.[c] The results are shown in figure 2.

Evidently the data sets on which the n-tuple method fails are those requiring astronomical numbers of hypercubes to cover the data. The exceptions are "Technical" which is covered by just 1 hypercube, and "DNA" which is unusual in that originally Boolean attributes were treated as integers.

It is not easy to fix this problem by tuning parameters. The "generalisation length" aK/n can be increased by increasing a or K, or decreasing n. These alterations on a and n run into difficulty with (2). Increasing K gives longer representations of the patterns, which may in turn require the use of more N-tuples to adequately sample them.

[c]Only the eigenvalues smaller than the edge length were rounded up; ie., they were dropped from the product. Neglecting to round the others does not affect the order of magnitude of the result.

5 Confidence intervals

The n-tuple system classifies a pattern to the class c that yields maximal tuple proximity (score) with discriminator D_c. The difference of maximal and next maximal score (the confidence interval) varies with the pattern. The mean confidence interval as a function of tuple size n is plotted in Figure 3A. The n-tuple system with 100 tuples was applied to the classification of several StatLog datasets. Two of them, Belgian II and Cut20, pose problems to the classifier (see section 3). The confidence interval increases with n, as higher-order correlations become available to the classifier. However, there seems to be no correlation between the size of the confidence interval and the percentage of correct classifications made.

Figure 3B presents the distribution of confidence intervals for the tuple size $n = 12$. It appears that the distributions tend to follow two forms: one approximately symmetrical, with a very low count of small confidence intervals, the other asymmetric with a considerable number of small score differences. The datasets on which the n-tuple classifier scores poorly seem to possess the symmetrical distribution.

This preliminary data seems to suggest, oddly, that the n-tuple classifier gives correct classifications with small confidence intervals, whereas mistakes are made "confidently".

6 Conclusions

A large set of comparative experiments shows that the n-tuple method is highly competitive with other popular methods, and other neural network methods in particular, except on data sets of high volume relative to the volumes naturally associated with the n-tuple method. Preliminary results suggest that confidence interval distributions fall into two categories, and that these are correlated with classification performance.

Acknowledgement

The authors are grateful to Louis Wehenkel of Universite de Liege for useful correspondence and permission to report results on the BelgianI and BelgianII data sets, Trevor Booth of the Australian CSIRO Division of Forestry for permission to report results on the Tsetse data set, and Reza Nakhaeizadeh of Daimler-Benz, Ulm, Germany for permission to report on the Technical, Cut20 and Cut50 data sets.

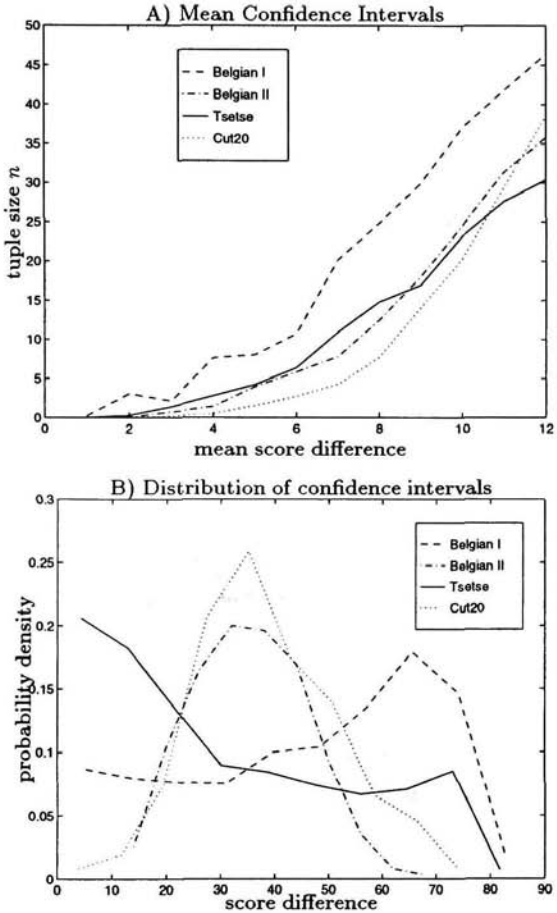


Figure 3: A) Average Confidence intervals as a function of tuple size n for several StatLog datasets. B) distribution of relative confidence intervals for tuple size $n = 12$.

References

1. N.M. Allinson and A. Kołcz. Distance relationships in the n-tuple mapping. submitted to Pattern Recognition.
2. N.M. Allinson and A. Kołcz. Enhanced n-tuple approximators. *Weightless Neural Network Workshop*, pages 38–45, 1993.
3. N.M. Allinson and A. Kołcz. Application of the cmac input encoding scheme in the n-tuple approximation network. *IEE Proceedings on Comput. Digit. Tech.*, 141(3):177–183, 1994.
4. W.W. Bledsoe and C.L. Bisson. Improved memory matrices for the n-tuple recognition method. *IRE Joint Computer Conference*, 11:414–415, 1962.
5. W.W. Bledsoe and I. Browning. Pattern recognition and reading by machine. *Proceedings of the Eastern Joint Computer Conference*, pages 225–232, 1959.
6. I. Tattersall S. Foster and R.D. Johnston. Single layer lookup perceptron. *IEE Proceedings-F*, 138(1):46–54, 1991.
7. R. Rohwer and D. Cressy. Phoneme classification by boolean networks. *Proceedings of the European Conference on Speech Communication and Technology*, 2:557–560, 1989.
8. R. Rohwer and M. Morciniec. The theoretical and experimental status of the n-tuple classifier. Technical Report NCGR/4347, Aston University Neural Computing Research Group, Aston Triangle Brimingham B4 7ET UK, 1995.
9. D. Michie D.J. Spiegelhalter and C.C. Taylor, editors. *Machine Learning, Neural and Statistical Classification*. Prentice–Hall, 1994.
10. I. Aleksander W.V. Thomas and P.A. Bowden. Wisard a radical step forward in image recognition. *Sensor Review*, pages 120–124, 1984.
11. J.R. Ullmann. Experiments with the n-tuple method of pattern recognition. *IEEE Transactions on Computers*, 18(12):1135–1137, 1969.
12. J.R. Ullmann and P.A. Kidd. Recognition experiments with typed numerals from envelopes in the mail. *Pattern Recognition*, 1:273–289, 1969.

COMPARISON OF SOME METHODS FOR PROCESSING 'GREY LEVEL' DATA IN WEIGHTLESS NETWORKS

R.J.MITCHELL, J.M.BISHOP, S.K.BOX AND J.F.HAWKER
Department of Cybernetics, The University of Reading,
PO Box 219, Whiteknights, Reading, RG6 6AY, UK

Weightless neural networks have been used in pattern recognition vision systems for many years. The operation of these networks requires that binary values be produced from the input data, and the simplest method of achieving this is to generate a logic '1' if a given sample from the input data exceeds some threshold value, and a logic '0' otherwise. If, however, the lighting of the scene being observed changes, then the input data 'appears' very different. Various methods have been proposed to overcome this problem, but so far there have been no detailed comparisons of these methods indicating their relative performance and practicalities. In this chapter the results are given of some initial tests of the different methods using real world data.

1 Introduction

The use of weightless networks for pattern recognition is well known and the first commercial hardware neural network vision system, WISARD[1], used weightless networks. These networks require binary values for forming the 'tuples' used to address their RAM neurons. In a simple system, the input to the network comprises a set of boolean values, and the tuples are formed by sampling these values. If however the input comprises 'grey level' data, the values used to form the tuples must be obtained by processing the input data in some manner.

The simplest method, 'thresholding', is to generate a logic '1' if the input value exceeds a specified threshold value, and a logic '0' if otherwise. One drawback of thresholding is that the tuples so formed can be very different when the d.c. level of the input data changes, which might occur in a vision system due to changes in the ambient light of a scene as viewed by a video camera.

This problem can be partly reduced by automatic thresholding, that is, by determining the threshold value of each input data. This can be achieved by automatic controls on a video camera, but if that option is not available suitable processing of the input data is required. Ideally the threshold value should be the median of the input data so that half of the data are logic '1'. However, calculating the median is relatively difficult, often involving sorting the data[2]. Therefore, as a compromise, the mean of the input data is often used, but this reduces the performance of the system.

One robust method of processing grey level data is to use 'thermometer coding'[3]. This involves converting each sampled point of the input into an array of Boolean values, where the greater the amplitude of the source, the more Boolean values are true. In a 4-bit thermometer code, the input data are quantised into five values, represented by 0000, 0001, 0011, 0111 and 1111; similarly, a 16-bit thermometer code quantises the data into seventeen values. This coding is equivalent to maintaining many threshold systems, each with a different threshold value. A major drawback of the method is that, for a t-bit thermometer code, t times as much memory is required for the RAM neurons than for thresholding.

Therefore, a different technique has been developed at the University of Reading, called Minchinton cells[4]. These cells are general purpose simple pre-processing elements which are placed between the input data and the RAM neurons and which can make the system more tolerant of changes in the d.c. level of the data without the need for any extra memory and for very little extra processing. The method has been used successfully, for example, in a hybrid weightless system which attempted to find the position of eyes within images of faces[5], and as part of an auto-associative weightless neural network[6].

Three types of Minchinton cells have been defined[4], all of which process values from the non-Boolean input data and which produce a Boolean result. In the following, let **I** be the input data and **I**[x] be the value at position x within these data: typically x is chosen randomly. The cell types are:

	Definition	**Function**
i)	**I**[x] > constant	Threshold
ii)	**I**[$x1$] > **I**[$x2$]	Type 0 cell
iii)	(**I**[$x1$] - **I**[$x1$ + 1]) > (**I**[$x2$] - **I**[$x2$ + 1])	Type 1 cell

The first of the above is thresholding, which is thus one form of the general Minchinton cell. As thermometer coding can be achieved with multiple thresholds, such coding can also be implemented using Minchinton cells.

The Type 0 cell returns a Boolean true if the value at position $x1$ in the input data exceeds that at position $x2$: this type gives better tolerance to changes in overall lighting for the following reason. Suppose the lighting of the input data increases. It is likely therefore, if saturation effects are ignored, that the values at the two positions will increase by about the same amount, hence the difference between the values will hardly change and so the output of the cell is likely to be unchanged. Any changes in the cell outputs that do occur can, of course. be compensated for by the generalisation

properties of a standard weightless recognition system.

The Type 1 cell simply compares two points in the input data by calculating the difference between adjacent points, each pair being chosen randomly. The effect of the Type 1 cell is to render the network insensitive to zonal d.c. changes in the input, such as shadows, except at zonal boundaries. However, as the Type 1 cell involves a second differential process, the network becomes more sensitive to high frequency changes in the image. This form of cell has not yet been used in practice.

The three types of Minchinton cell can, like weightless networks, be easily implemented in hardware, for example, by using the modular hardware system developed at Reading[7].

Another potential advantage to the Minchinton cell is that it uses more information than the other methods. If simple thresholding is used and, for example, the input data are 8-bit values and the threshold is 100, the system cannot distinguish between an input value of 120 and one of 160. Thermometer coding effectively provides more threshold values, so this problem is reduced, but only slightly. With the Type 0 Minchinton cell, each input data value is compared with another data value, not a constant, so overall the system may be able to detect higher frequency information than the other methods. This can become more significant when the input data are oversampled, that is, each value from the input space is sampled more than once; then each sample is likely to be compared with many different values.

Although these Minchinton cells have been used successfully, little research has as yet been done to compare their performance with that of thresholding or thermometer coding. The purpose of this chapter, therefore, is to report on such comparisons. These comparisons are achieved by testing images from a video camera, looking at the success of the network is terms of pattern recognition and discrimination. The tests were done on weightless networks which employed thresholding, thermometer coding and the Type 0 and Type 1 Minchinton cells.

2 The Tests

2.1 Test Data

The aim of the research is to investigate the response of weightless networks in recognising and discriminating different objects which are subject to changes in how they are illuminated over time. Although software simulations could be used, it was felt that using real world data would add verisimilitude to the work. As vision systems are often used on production lines, it was decided to choose some suitable products and record their appearance under different lighting conditions using a standard video camera. The products chosen were four disk boxes as they were convenient and of appropriate shape and size.

These boxes were placed under a video camera such that all four were visible at

one time and then the image from the camera was captured every 10 minutes over a 24 hour period and stored for later processing. Each complete image comprised 512×512 8-bit values, so the image of each disk box was of size 256×256 bytes. Then, when each image was tested, the data for the four disk boxes were separated and processed by VISIWIN[8] a Windows™ based product in which various configurations of weightless networks can be simulated.

Figure 1 shows four of the images of data used in the tests. Each image comprises the four diskette boxes, and each image was taken at a different time of the day. As can be seen from these images, the system is being asked to recognise images subject to different amounts of illumination and to shadows. In the tests four discriminators were used: the Fuji MF2HD box was taught in the first discriminator, the 3M box to the second, the Fuji MF2DD to the third, and the Dysan box to the fourth.

Figure 1: Four images, each of four diskette boxes, taken during the 24 hour period

2.2 *Weightless networks tested*

Nine network configurations were tested, these were:

Thresholding, where the threshold was the mean of each data input

4-bit Thermometer coding
16-bit Thermometer coding
Type 0 Minchinton cell with no oversampling
Type 0 Minchinton cell with 4 times oversampling
Type 0 Minchinton cell with 16 times oversampling
Type 1 Minchinton cell with no oversampling
Type 1 Minchinton cell with 4 times oversampling
Type 1 Minchinton cell with 16 times oversampling.

For each network type, the software was configured with four discriminators and images from each of the four disk boxes were trained in the associated discriminator.

The full range of results will be reported elsewhere. Here however a summary of the important results will be given. For these tests a suitable tuple size was chosen, namely 8, each network was trained on a few images, and then the response of each discriminator was recorded. This process was performed twice: in test A the training data were a few images at the start of the sequence; in test B the training data were images taken once each hour during the day.

3 Results

A multi discriminator system seeks to recognise and discriminate data. The best measure for this purpose is relative confidence[9], defined as follows, where the response of a given discriminator is the number of RAM neurons which report a '1' when the data are presented:

$$\frac{\text{Response of best discriminator - Response of next best discriminator}}{\text{Response of best discriminator}}$$

Note, where a network misrecognised the input image, the relative confidence measure was set to 0.

In the tests done, the three Type 0 Minchinton Cell networks responded almost identically, as did the three Type 1 Minchinton Cell networks, and the 4-bit thermometer code network was significantly better than the 16-bit thermometer code network. The graphs below thus show the relative confidence of the four discriminators for the network using thresholding, the 4-bit thermometer code network, and the Type 0 and Type 1 Minchinton cells with no oversampling.

Figure 2 shows graphs of Relative Confidence for the four discriminators when the networks were trained on data at the start of the sequence, that is test A; and figure 3 shows the responses of the discriminators when the network was trained on data once every hour, that is test B.

4 Discussion

The graphs clearly show that the Type 0 and Type 1 Minchinton cells are better than thermometer coding which itself is better than thresholding. Usually the Type 0 cell is better than the Type 1 cell, however, in some of the graphs in test B, the Type 1 cell performs best. In test B, the network is trained on more representative data: thus it seems that the Type 0 cell is better at generalisation.

As mentioned earlier, the 4-bit thermometer code has a better relative confidence than the 16-bit code. This is because the response of all discriminators using the 16-bit thermometer code were consistently higher than those when the 4-bit code was used and hence the relative difference between the 'best' and the 'next best' discriminator is smaller.

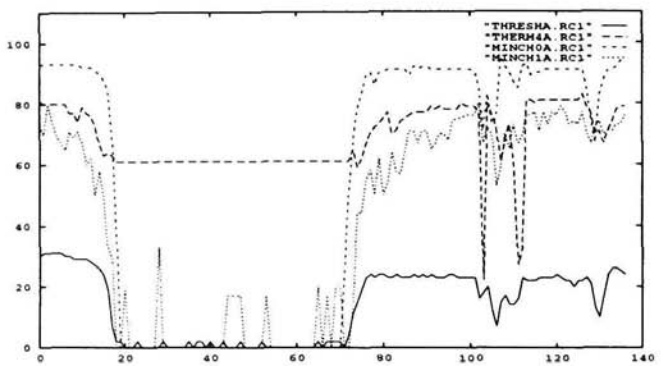


Figure 2a: test data were the 'Fuji MF2HD'

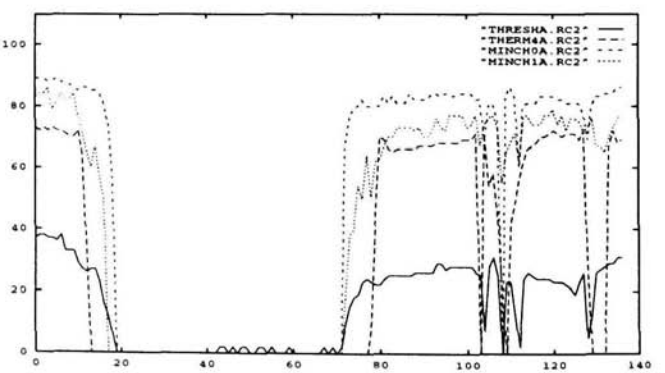


Figure 2b: test data were the '3M'

Figure 2: Relative Confidence for four discriminators for Test A

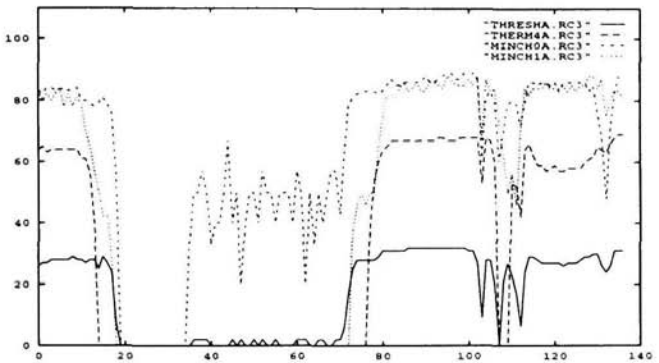


Figure 2c: test data were the 'Fuji MF2DD'

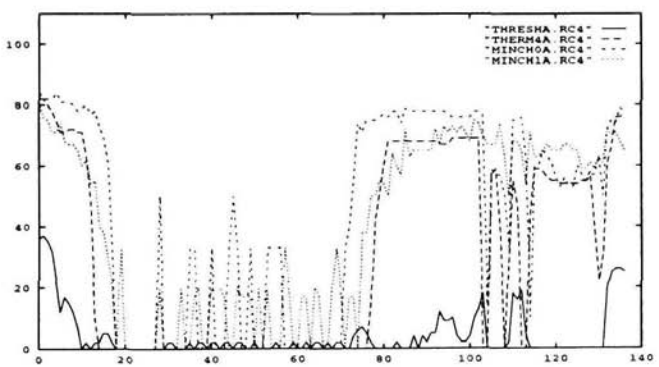


Figure 2d: test data were the 'Dysan'

Figure 2: Relative Confidence for four discriminators for Test A

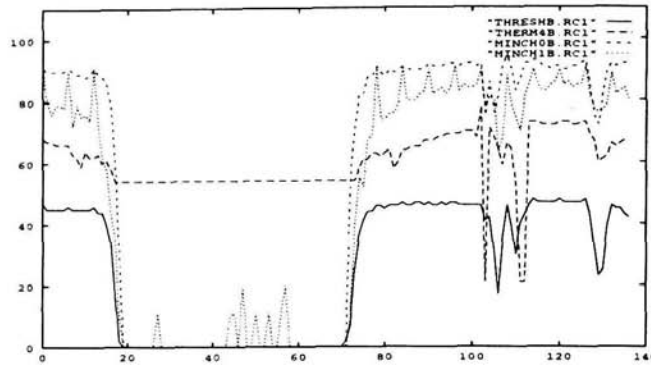


Figure 3a: test data were the 'Fuji MF2HD'

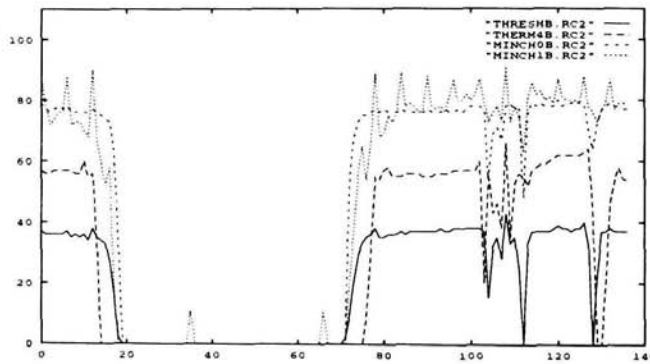


Figure 3b: test data were the '3M'

Figure 3: Relative Confidence for four discriminators for Test B

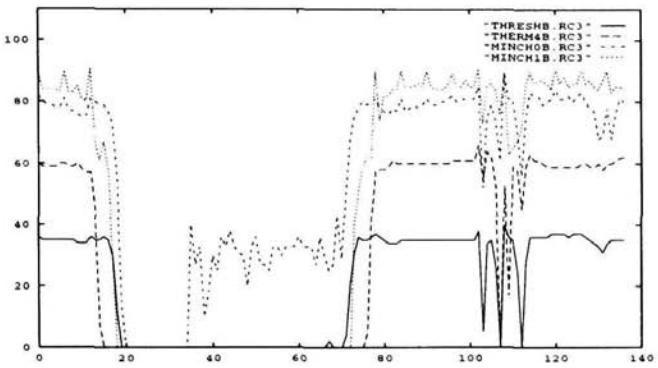


Figure 3c: test data were the 'Fuji MF2DD'

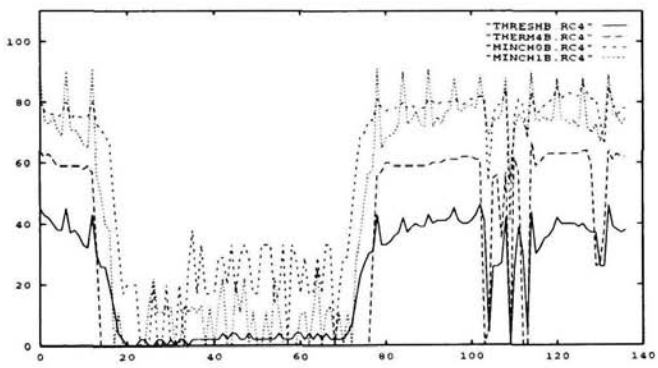


Figure 3d: test data were the 'Dysan'

Figure 3: Relative Confidence for four discriminators for Test B

The period on each graph labelled between 20 and 70 is at night, when the disk boxes were illuminated very poorly. As one would expect, the performance of all networks was reduced. However, Thermometer coding performed well on image 1 because most of it was dark.

In the period between 100 and 110, significant shadows occurred in the images,

which is the cause of the sharp dips in the performance of all networks. On the A set of data, the Type 0 cells performed best over this interval; on the B set, the Type 1 cells were slightly better.

5 Conclusion

These results clearly demonstrate that the best method for pre-processing continuous data for use by a weightless network is the Type 0 Minchinton cell. In general this gives higher relative confidence, generalises better and is both more computationally and memory efficient than the other methods investigated.

References

1. I. Aleksander, W.V. Thomas. and P.A. Bowden (1984) 'WISARD: a radical step forward in image recognition', *Sensor Review*, pp120-124.
2. N.Wirth (1986) 'Algorithms and Data Structures', Prentice-Hall.
3. I. Aleksander and R.C. Albrow (1968) 'Pattern Recognition with AdaptiveLogic Elements', IEE-NPL Conf. Pattern Recognition.
4. J.M. Bishop, P.R. Minchinton & R.J. Mitchell (1991) 'Real Time Invariant Grey Level Image Processing Using Digital Neural Networks', Proc IMechE Conf, EuroTech Direct '91 -Computers in the Engineering Industry, pp187-199, Birmingham.
5. J.M. Bishop and P. Torr (1992) 'The Stochastic Search Network', in 'Neural Networks for Vision, Speech and Natural Language', Ed. D.J.Lingard and C.Nightingale, pp 370-387.
6. J.M. Bishop and R.J. Mitchell (1995) 'Auto-associative memory using n-tuple techniques', Intelligent Systems Engineering, Vol 3, No 4, pp 222-229.
7. R.J. Mitchell, P.R. Minchinton, J.M. Bishop, J.W.Day, J.F.Hawker and S.K.Box (1993) 'Modular Hardware Weightless Neural Networks', Proc. WNNW'93, pp123-128, University of York.
8. Cybertronix Ltd (1995) 'VISIWIN'.
9. J.M. Bishop (1989), 'Anarchic Techniques for Pattern Classification', PhD Thesis, University of Reading, UK.

A FRAMEWORK FOR REASONING ABOUT RAM-BASED NEURAL NETWORKS FOR IMAGE ANALYSIS APPLICATIONS

G. HOWELLS, D.L. BISSET, M.C. FAIRHURST
Electronic Engineering Laboratories,
University of Kent, Canterbury, Kent, CT2 7NT, UK

A Framework is introduced for reasoning about RAM-based weightless neural networks using networks created using the Goal-Seeking Neuron (GSN) as an example. The framework is then generalised to allow it to encompass all architectures within the RAM-based neural network paradigm. The framework may form the basis of a simple logic for RAM-based networks allowing formal comparison of the performance of various architectures and the development of provably optimal solutions within given constraints.

1 GSN

Many practical pattern recognition problems may be solved by performing operations on simple thresholded images. It can be useful to visualise such pattern recognition problems as a mapping from a binary input pattern to a given binary output pattern. In such cases, pattern recognition is reduced to a Boolean or logic problem with the inputs and outputs expressed using vectors of two valued components. RAM-based weightless neural networks are a type of neural network well suited to the expression of solutions to such logical problems. The Goal Seeking Neuron may be employed to construct such architectures.

The Goal Seeking Neuron (GSN) is a RAM-based neuron where the inputs and output of the neuron are taken from the values "0", "1" and the undefined value "u". The inputs to a neuron form an addressable set incorporating all memory locations which may be formed by treating any undefined value within the input as either a "0" or a "1". The output of a neuron will be the defined value which occurs most often within the memory locations included within the addressable set. If the addressable set contains either no defined value or an equal number of both defined values, then the undefined value "u" is output.

Pyramid networks of GSN with multiple pyramids may be constructed by assigning a desired output for each pyramid associated with each pattern learnt by the network. Learning requires training the network with values mapping the input associated with the pattern with the desired output (if possible). For recognition, each pyramid behaves independently for the area of the sample pattern to which it is

clamped. Each neuron maps its input values to its output values as described above. Recognition occurs when the outputs of all pyramids coincide with the desired outputs for a given stored pattern. Detailed descriptions are given in[2].

2 Formalising GSN

The primary idea of the formal framework is to associate pattern identifiers with the different processes associated with GSN network operations. In such a way, it becomes clear which patterns are under consideration by which neurons at any one time during network operation with the eventual emergence of a single unique pattern indicating successful recognition of that pattern by the network.

Initially, an individual network node is considered. Since there are only two defined values ("0" and "1") which may be output by a GSN, each pattern learnt by the network will cause one of these two values to be produced. It is thus possible to consider the output of a neuron to be one of two sets (called discrimination sets), one containing all the patterns which cause a "0" to be produced, and the other containing all the patterns which cause a "1" to be produced.

During recognition, a pattern may be presented to the neuron which does not address a location containing either defined value. In this case an undefined value `u' is produced by the neuron. This is analogous to both discrimination sets being output by the neuron. Note that this is not the same as any arbitrary pattern being produced by the neuron. In the next layer of the pyramid the "u" will be interpreted as being either a "0" or a "1" when the addressable set is calculated. This is analogous to using either of the two discrimination sets of the neuron, not any arbitrary set of patterns. The output of a single neuron may thus be modelled as a set of sets containing one or two elements. The constituent sets will be one or both of the discrimination sets for a neuron.

For a first layer neuron, a pattern presented for recognition will either correspond exactly with a number of stored patterns in the region under consideration by the neuron (two or more patterns may be identical within a given region, especially if neuron connectivity is low) or fail to match any stored pattern. The input to a neuron thus may again be considered to be a (possibly empty) set of patterns. In order to distinguish which of the two discrimination sets for the neuron should be considered as relevant to the input set, it is necessary to split each of the discrimination sets into subsets. Each subset will be analogous to a memory location and the number of elements within a discrimination set will be equal to the number of memory locations containing the symbol associated with the discrimination set. The output of the neuron will be the discrimination set (with its elements flattened into a single set) of which the input set of the neuron is an element. If the input set is not a member of either set, both are output (a "u" will have been referenced).

Second and subsequent pyramid layers are presented with a set of patterns from each input neuron. The input set for a given neuron will be the intersection of the out-

put sets of each input neuron. The output is found, as above, by the discrimination set of which the input set is a member. (To understand this, think of a neuron of connectivity two where one input identifies the pattern as A or B and the other identifies the pattern as B or C. The input set will be the pattern B.)

A further operation is required if both discrimination sets from a previous layer neuron have been presented as input (a "u" was output). It is necessary here to form the cartesian product of the input sets before taking the intersection of each element of the product---analogous to each alternative address being tried if a "u" exists on the input. The intersection will now be a set of sets and the discrimination set produced by the neuron will be the one sharing most common elements with the intersection set--this corresponds to the greater of the number of "0"s and "1"s within the addressable set.

This simple framework allows reasoning to be performed on GSN networks simply using operations on sets of patterns.

3 Generalisation

The above model allows precise reasoning to be performed about GSN networks. With some abstraction however, the model may be generalised to allow reasoning about all RAM-based neural network architectures (for examples see[1]).

The model consists of a formal mathematical framework for RAM-based neural network architectures which for the first time has enabled such architectures to be formally analysed and compared. The framework has already proved invaluable both in giving an intuitive understanding of the operation of existing architectures (something which often proves opaque when considering neural networks) and in aiding the development of new architectures. In the latter case, aspects of the behaviour of a new network architectures may be accurately predicted without the need for possibly lengthy architecture construction or simulation.

At the intuitive level, the theory operates by manipulating sets of patterns in such a way as to associate each possible neuron address location and output with a set of patterns (either training or target) which the particular contents of the location or output represents (for example, if neuron x produced an output y, the theory would give meaning to symbol y by associating it with, again for example, the pattern classes with which it was consistent, that is, output y means the neuron has recognised' the given set of pattern classes). Each neuron is also associated with discrimination sets which define sets of patterns between which it is unable to distinguish. The theory further associates the sets of patterns associated with the memory locations of a neuron with the discrimination sets for the neuron.

The sets employed by the theory (which also acts to define a RAM-based neural network) are defined as follows:-

- a set of neurons $\mathcal{N}$ where each n $\in \mathcal{N}$ is a pair of the form $< i, \mathcal{S}_n >$ where i is a unique natural number and $\mathcal{S}_n$ is defined to be:-

- a set of symbols where each element s $\in \mathcal{S}_n$ is of the form $<\mathcal{M}s,\{\mathcal{D}_1...\mathcal{D}_{sx}\}>$ where sx represents the number of discrimination sets associated with symbol s and:-

- $\mathcal{M}_s$ is a set of pairs $< i, \{p_1...p_{mx}\} >$ where mx represents the number of patterns having addressed location $m \in \mathcal{M}$, i is a unique natural number and $p_1...p_{mx}$ are taken from the set of patterns $\mathcal{P}$

- Each element of the discrimination sets $\mathcal{D}_1...\mathcal{D}_{sx}$ is also a pattern taken from the set of patterns $\mathcal{P}$ where

- $\mathcal{P}$ is the union of the sets $\mathcal{P}^I$ and $\mathcal{P}^C$, the sets of training and target patterns respectively.

- Connections between neurons are represented by a set C where $c \in C$ is a pair of the form $< n_o, n_i >$ where $n_o \in \mathcal{N}$ or is taken from a set of virtual input neurons I whose cardinality is equal to the number of symbols comprising the sample patterns and $n_i \in \mathcal{N}$ or is taken from a set of virtual output neurons O where the cardinality of O is network dependent. $i \in I$ or $i \in O$ is a natural number used to index the network input and output.

- Architecture dependent features are defined via the functions **IN**, **ADDRESS** and **COMPRESS** to model the input, pre-neuron processing and post-neuron processing respectively.

Note that the set of symbols is expressed by the relationship between memory locations and discrimination sets---no explicit symbols are employed. Figure 1 provides a pictorial representation of the structure of the neuron set which should aid understanding.

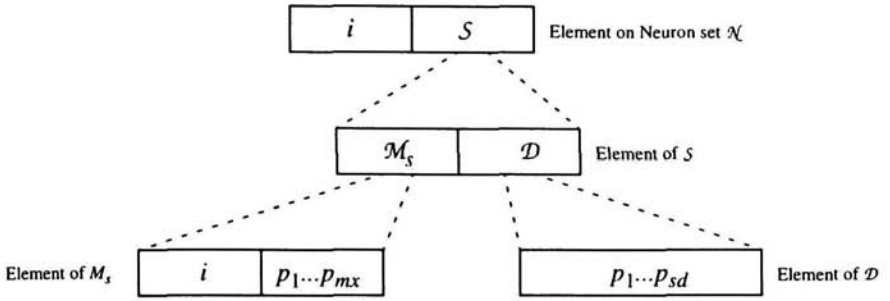


Figure 1: Formal model for the set of neurons $\mathcal{N}$

The above structures are employed to represent a network having been presented with a particular training set (rules for which are not included here). Further sets may be used to model the behaviour of the network during recognition. A formal set of rules defining the conditions necessary for membership of the various sets employed to model recognition given the definition of the network introduced above are now presented:-

(network-input) $i \in N$, **IN**, *sample* $\vdash$ <i, **IN**, $sample_i$> $\in$ OUT

where $sample_i$ is the i'th symbol of the sample pattern *sample* and OUT is the set of neuron outputs whose elements are pairs of the form $< n, \{\mathcal{D}_1 \ldots \mathcal{D}_x\} >$ where $n \in \mathcal{N}$ or $n \in I$ and $\mathcal{D}_1 \ldots \mathcal{D}_x$ are the discrimination sets produced by neuron n as output (x may vary depending on the symbol produced by the neuron - recall that differing symbols may produce a differing number of discrimination sets).

The rule thus merely states that the output of a virtual input neuron is defined by applying the function **IN** to the corresponding symbol to the neuron within a sample pattern presented to the network.

The inputs to a particular neuron $n \in \mathcal{N}$ within the network may be deduced via the following rule:-

(neuron-input) $<n', n> \in C$, $<n, \{\mathcal{D}_{n'1} \ldots \mathcal{D}_{n'x}\}> \in OUT \vdash \{\mathcal{D}_{n'1} \ldots \mathcal{D}_{n'x}\} \in IN_n$

where $I\mathcal{N}_n$ is the set of output discrimination sets of neurons providing inputs to neuron n.

The address set $\mathcal{A}_n$ of a neuron $n \in \mathcal{N}$ may be deduced by the following rule using the architecture dependent function **ADDRESS**:-

$$\textbf{(address)}\ I\mathcal{N}_n,\ \textbf{ADDRESS} \vdash \mathcal{A}_n \equiv \textbf{ADDRESS}\ I\mathcal{N}_n$$

The relationship between the outputs of neuron $n \in \mathcal{N}$ and the address set $\mathcal{A}_n$ may be deduced by:-

$$\textbf{(recall)}\ a \in \mathcal{A}_n,\ <i,\frown a> \in \mathcal{M}_s,\ <\mathcal{M}s,\ \{\mathcal{D}_{n1} \ldots \mathcal{D}_{ns}\}> \in \mathcal{S}_n \vdash \{\mathcal{D}_{n1} \ldots \mathcal{D}_{ns}\} \in \mathcal{PAR}_n$$

where $\mathcal{PAR}_n$ represents the set of partial solution produced by neuron n for each $a \in \mathcal{A}_n$ (to understand this rule, compare with calculating set intersections in GSN).

The total (or complete) output for a neuron $n \in \mathcal{N}$ may be deduced from the partial solutions using the definition of the **COMPRESS** function as follows:-

$$\textbf{(compress)}\ \mathcal{PAR}_n,\ \textbf{COMPRESS},\ \mathcal{OUT} \setminus <n,x> \vdash <n,\textbf{COMPRESS}\ \mathcal{PAR}_n> \in \mathcal{OUT}$$

The rule states that the total output is found by applying **COMPRESS** to the set of partial outputs. Note that any existing output for the neuron $<n,x>$ must be ignored. The set $\mathcal{OUT}$ may only contain one element for each neuron.

The network outputs may be deduced from the set of outputs $\mathcal{OUT}$ by looking at inputs to the virtual output neurons $i \in O$.

The deduction of the sets of patterns above allows proofs about the behaviours of given networks to be constructed. The sets also help to give a more intuitive understanding of the meaning of various network outputs.

4 Conclusions

A framework has been introduced to allow reasoning about RAM-based neural networks. The framework allows:-

- the efficiency of various network architectures to be compared by formal

reasoning rather than by comparison of statistical results,

- the optimality of a particular architecture to be determined,
- the relative optimality of various components of a given architecture to be compared
- an intuitive understanding of the meaning of partial and intermediate network results to be given (via the sets of patterns that such results represent),
- the possibility of providing proofs that a given architecture will behave within certain pre-defined constraints.

Note that the framework abstracts away from many architecture features. For example, whether neurons are arranged in regular structures such as layers or pyramids plays no part in the framework. All information regarding neuron connections is contained within the set C whether these connections form a regular structure or not.

References

1. N. M. Allinson, (editor) Proceedings of the Weightless Neural Network Workshop, *University of York,* April 1993.
2. E.C.D.B.C Filho, M.C. Fairhurst, D.L. Bisset, Adaptive Pattern Recognition using goal seeking neurons, *Pattern Recognition Letters*, 12:131-138 1991.
3. E.C.D.B.C Filho, D.L. Bisset, M.C. Fairhurst, Architectures for Goal Seeking Neurons, *Journal of Intelligent Systems*, Vol~2:95-119 1991.

CROSS-VALIDATION AND INFORMATION MEASURES FOR RAM BASED NEURAL NETWORKS

T.M. JØRGENSEN, S.S. CHRISTENSEN and C. LIISBERG
Risø National Laboratory, P.O. Box 49,
DK-4000 Roskilde, Denmark

It is shown that it is simple to perform a cross-validation test on a training set when using RAM based Neural Networks. This is relevant for measuring the network generalisation capability (robustness). An information measure combining an extended concept of cross-validation with Shannon information is proposed. We describe how this measure can be used to select the input connections of the network. The task of recognising handwritten digits is used to demonstrate the capability of the selection strategy.

1 Introduction

When an artificial neural network (NN) is being trained to a given task, one would like to achieve a robust network in the sense that the performance of the network is relatively unaffected by redrawing a specific example from the training set. If one counts the number of examples in the training set that can be correctly classified by training the network on the remaining examples, it corresponds to the so-called leave-one-out cross-validation (CV) test[1]. The error rate obtained with a cross-validation procedure on the training set is a nearly unbiased estimator of the test error[2]. Accordingly, as the size of the training set increases the mean value of the cross-validation error will approach the mean of the test error. It follows that when optimising a neural network architecture one could try to optimise the leave-one-out cross-validation error. It is however not simple to perform a leave-one-out cross-validation test on a conventional feed-forward NN architecture, as it is computationally expensive due to replicated training sessions[3].

If one uses a RAM-based NN[4,5] we illustrate in this chapter that it is straightforward to perform a leave-one-out cross-validation test. By selecting input connections based on CV error one obtains a network with a better generalisation performance than a network of the same size trained with random connections. Instead of counting the number of misclassifications in a CV test a more informative measure is the average information (in the Shannon sense) obtained per example by a leave-one-out crossvalidating test. We describe how it is possible to calculate the average mutual information gained from the network in a leave-one-out CV test.

Often there will exist severe differences between the distribution of the training set and the "real" distribution, especially when the training set size is small. In such

situations the CV error can be a poor estimator of the test error. A potential problem with a leave-one-out crossvalidation test is the following. If each pattern of the training set occurs more than once, the CV-error for a RAM based NN will be 0 (ignoring ambiguous decisions). Since it is very possible for many recognition tasks that specific examples occur more than once, it is necessary to deal with this problem. We have therefore extended the above mentioned information measure to deal with a more robust cross-validation concept. We have denoted the information measure cogentropy (as it COmputes the GENeralisation capability by use of enTROPY). We show that use of the cogentropy measure solves the problem of having multiple occurrences of the same example in the training set.

In order to evaluate the applicability of the cogentropy measure we have tested its use on the task of recognising hand-written digits.

2 Training procedures for RAM based neural nets

2.1 The conventional training strategy

The RAM based neural network can be considered as a number of Look Up Tables (LUTs). This is illustrated in Fig. 1. Each LUT probes a subset of the binary input data. In the conventional scheme the bits to be used are selected at random. The sampled bit sequence is used to construct an address. This address corresponds to a specific entry (column) in the LUT. The number of rows in the LUT corresponds to the number of possible classes. For each class the output can take on the values 0 or 1. A value of 1 corresponds to a vote on that specific class. The output vectors from all LUTs are added, and subsequently a winner takes all decision is made to perform the classification. In order to perform a simple training of the network the output values are initially set to 0. For each example in the training set the following steps are then carried out:

- Present the input and the target class to the network.
- For all LUTs calculate their corresponding column entries.
- Set the output value of the target class to 1 in all the "active" columns.

By use of this training strategy it is guaranteed that each training pattern always obtains the maximum number of votes. As a result the network makes no misclassification on the training set (ambiguous decisions might occur).

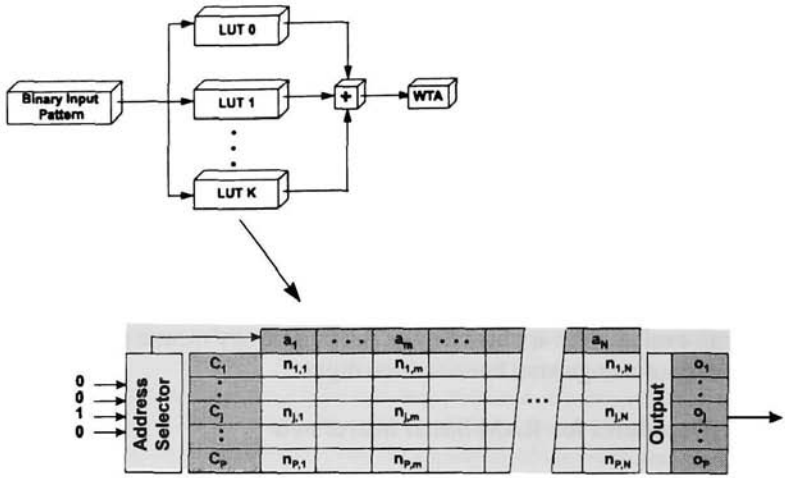


Figure 1: Illustration of the RAM-based neural network structure. The number of training examples from class c_j that generate the address a_m is denoted $n_{j,m}$. If $n_{j,m} \geq 1$, the output value, o_j, is set to 1, else 0.

The generalisation capability of the network is directly related to the number of input bits for each LUT. If a LUT samples all input bits then it will act as a pure memory device and no generalisation will be provided. As the number of input bits is reduced the generalisation is increased at an expense of an increasing number of ambiguous decisions. Furthermore, the classification and generalisation performances of a LUT are highly dependent on the actual subset of input bits probed. The purpose of an "intelligent" training algorithm is thus to select the most appropriate subsets of input data.

2.2 *Cross-validation*

The idea of a cross-validating procedure is to test whether each of the training patterns can be learned by the NN if it is trained on the remaining patterns. During the normal recall process of the LUT contents one obtains an output value of 1 if the corresponding column entry is visited by one or more training patterns. In order to perform the CV-test it is necessary to know the actual number, n_{jm}, of training examples that have visited the cell corresponding to column a_m and class c_j. Accordingly, these numbers are stored in the LUTs (Fig. 1). A CV-test is now performed in the following manner:

- Present the input and the target class c_k to the network.

- For all LUTs calculate their corresponding column entries a_m.
- If ($n_{j,m} > 1$ and $j = k$) or ($n_{j,m} > 0$ and $j \neq k$) then set the output value, o_j, of the target class, j, to 1.
- Calculate the winner class of all LUTs.
- If the class obtaining most votes differs from the target class, then increment the CV-error rate.

A simple input selection strategy that makes use of a CV-test is as follows:

- N LUTs with chosen input sets (possibly random) are trained.
- A CV-test is performed on each LUT.
- The best LUT (low CV-error) is chosen as the first LUT (LUT 0) of the network.
- A new set of N LUTs is trained.
- N CV-tests are performed by combining each of the new N LUTs with LUT 0.
- The best LUT (low CV-error) is chosen as the second LUT (LUT 1) of the network.
- The procedure is repeated until a satisfactory CV-error is obtained or until the CV-error saturates.

When selecting the input connections for the LUT candidates it can often be advantageous to restrict the possible connections in different ways. One way is to force different LUTs to pick connections from subspaces obtained from principal component analysis.

The advantage of the above described selection procedure is that one obtains a network with a better generalisation performance than a network of the same size trained without a selection scheme. In addition the use of a cross-validation procedure

makes it possible to estimate the number of input connections to be used pr. LUT. The determination of the number of input connections corresponds to evaluating the appropriate number of hidden units in a conventional feed-forward architecture. This stems from the fact that the number of LUT columns increases with the number of inputs, so that the number of columns with non-zero values (equivalent to hidden units) increases too. For a conventional feed-forward net it is quite difficult with basis in the training examples to determine the number of hidden units to use[6]. As a leave-one-out cross-validation test is easy to perform on RAM based nets the corresponding task becomes much simpler for these architectures.

A problem with a leave-one-out CV-test is the following. If each pattern of the training set occurs more than once the CV-error for the RAM based NN will be 0 (ignoring ambiguous decisions). Since it is very possible for many recognition tasks that specific examples occur more than once, it is preferable to deal with this problem. The cogentropy-measure described in the following section is able to handle this problem.

2.3 The cogentropy measure

In the following we will describe how the average mutual information gained from the network in a leave-one-out CV-test can be calculated. Then we extend the measure to deal with a more robust cross-validation concept. We have denoted the extended measure cogentropy (as it COmputes the GENeralisation capability by use of the information enTROPY).

The mutual information $I(a_m,c_j)$ between the column entry a_m and the class c_j is defined as follows[7]:

$$I(a_m, c_j) = \log\left(\frac{p(c_j|a_m)}{p(c_j)}\right) \tag{1}$$

where $p(c_j|a_m)$ is the probability that an object generating the column entry a_m belongs to the class c_j. $p(c_j)$ is the probability that an object belongs to class c_j. In order to estimate these probabilities it is tempting to use the feature distribution of the training set:

$$p(c_k|a_m) = \frac{n_{km}}{\sum_j n_{jm}}, \quad p(c_k) = \frac{\sum_m n_{km}}{\sum_{jm} n_{jm}} \tag{2}$$

However, for these estimates to be reliable a large and representative training set

is required. In reality it is hard to determine whether a training set is representative or not. Furthermore, one would like to operate on small training sets. Yet if one decides to use these estimates there is a great risk of putting to much emphasis on the distribution of features in the training set. A much more robust system is to use estimates based on the following two arguments:

- If a given feature exists in k classes of out P in the training set, then there is certainly a probability greater than zero for this feature to exist in each of these k classes. However, since the actual distribution obtained on the training set cannot be associated with the general distribution, we attribute to each of the k classes an equal probability of having the actual feature (rectangular distribution). In this way we do not a priori favour any of the k classes with respect to this specific feature.

- If a given feature does not exist in a specific class in the training set, then the possibility exists that this feature never appears for objects belonging to this class. In any case, we have no instance in support of claiming that there is a probability larger than zero for objects belonging to this class to possess the given feature. Accordingly, we associate a small probability ε ($\varepsilon \ll 1$) for these cases.

These two arguments explain why it is sensible to let the LUT cell output values, o_j, be either 0 (ε) or 1. According to these arguments the estimate of $p(c_j|a_m)$ becomes:

$$p(c_j|a_m) = \begin{cases} \dfrac{1}{\sum_j \Theta(n_{jm})}, & \text{if } n_{km} > 0 \\ \varepsilon, & \text{if } n_{km} = 0 \end{cases} \tag{3}$$

$$\text{where } \Theta(x) = \begin{cases} 1, & \text{if } x \geq 1 \\ 0, & \text{if } x = 0 \end{cases}$$

Considering the classes to be equally likely (corresponding to no a priori information) we obtain for $p(c_j)$:

$$p(c_j) = \frac{1}{P} \tag{4}$$

Consider now the case where an example has been extracted from the training set. By use of Eq. (1) we want to calculate the amount of information that can be obtained about this specific example by using a specific LUT. Two cases must be considered:

- $n_{jm} > 1$ before extraction: In this situation the values of the probabilities defined in Eq. (3) do not change by extracting the considered example.

- $n_{jm} = 1$ before extraction: In this case the extraction of the example will change the values of the probabilities defined in Eq. (3). Accordingly, one obtains a wrong classification unless all other output values are zero. In order for formula (1) to deal with this situation it is necessary to introduce a concept of negative information. The information loss is given by the information needed to obtain a fully ambiguously classification (corresponding to zero information). The details of this concept are outlined in Jørgensen[8].

By extending the above ideas to deal with combination of LUTs one can calculate the average information contained in the network about a training example. The details of this will be described elsewhere.

In order to increase the robustness of our information measure we now consider a generalised type of cross-validation. Let I^b denote the information gained about a specific example by making a winner takes all classification. Now for each example one can calculate what we denote the *critical* number of examples. The critical number of examples is defined as the smallest number of examples (belonging to the same class as the training example in question) that (by an intelligent selection) must be redrawn from the training distribution in order for the classification to become false or ambiguous. The critical number, n_{crit}, is a measure of the robustness of the classification. The example information contained in the network after removal of the critical examples is denoted I_a. The information loss obtained by extracting the critical examples from the training set can be expressed as:

$$I_{loss} = I_b - I_a \tag{5}$$

The average information loss per critical example becomes:

$$I_{averageloss} = \begin{cases} \dfrac{I_{loss}}{n_{crit}}, & \text{if } n_{crit} > 0 \\ 0 \quad , & \text{if } n_{crit} = 0 \end{cases} \tag{6}$$

Now the information gained in a generalised crossvalidating test is defined as:

$$I_{genCV} = \begin{cases} I_b - I_{averageloss} = \left(1 - \frac{1}{n_{crit}}\right) I_b + \frac{I_a}{n_{crit}}, & \text{if } n_{crit} > 0 \\ I_b = I_a, \text{ if } n_{crit} = 0 \end{cases} \quad (7)$$

It is noted that when classification of a training example is considered, the example to be classified is itself a critical one.

The advantage of using equation (7) as optimisation measure instead of the CV error is due to the fact that it takes the amount of supporting examples into account. Four supporting examples counts more than one supporting example when Eq. (7) is used, whereas the conventional leave-one-out CV error does not distinguish between these two cases. Accordingly, the cogentropy will not be "cheated" if a training example is represented twice.

Unfortunately, it is impracticable to estimate the true n_{crit}-values when the number of training examples is large. It is however fairly easy to obtain a worst case (and operational) estimate by making use of the n_{jm} numbers. When calculating the n_{crit}-values for test examples they act as a confidence measure.

3 Results

In order to illustrate the capabilities of the selection strategies we have chosen the task of recognising hand-written digits. The digits used are from the NIST database[9]. The digits were scaled into a 16x16 format and centred.

The appropriate number of input connections pr. LUT can be determined by performing a leave-one-out cross-validation test on the training set. A system containing 50 LUTs is trained with an increasing number of input connections pr. LUT. For each case the cross-validation error is calculated. The minimum of the resulting performance curve is normally rather flat (see Fig. 2) allowing the number of input connections to vary within this range.

Several RAM nets were composed by selecting LUT connections based on the cogentropy measure or the cross-validation error rate. Table 1 compares these results with the results obtained by picking all connections by random. The decrease in error rate by using the cogentropy is also noted. The decrease from 4.0% to 3.5% might seem small, but it is actually the small percentages of errors that are difficult to avoid.

Table 1 shows that the use of the cogentropy measure reduces the required number of LUTs considerably for a given performance demand. It is seen that 100 LUTs selected by using the cogentropy measure performs better than 807 LUTs chosen randomly (507 of these LUTs were restricted to pick their connections in varying local receptive fields of size 4×4). As the recall time and the memory requirements are proportional to the number of LUTs in the system this would speed up the recall

time of the system by a factor of 8 and, furthermore, reduce the memory requirements with a factor of 8.

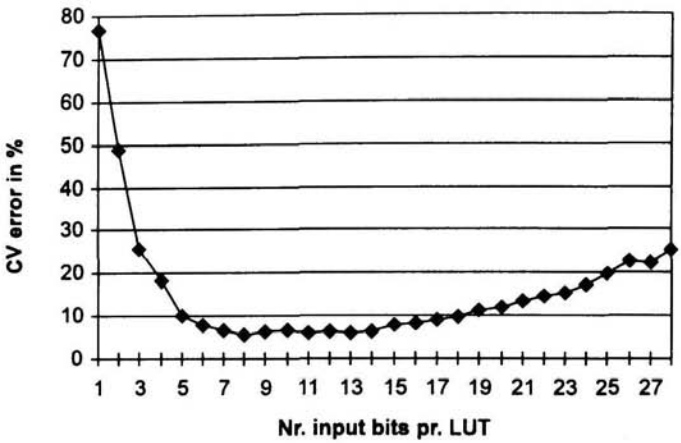


Figure 2: Variation of cross-validation error with the number of input bits sampled per LUT. The RAM nets trained consisted of 50 LUTs and was trained on 500 handwritten digits.

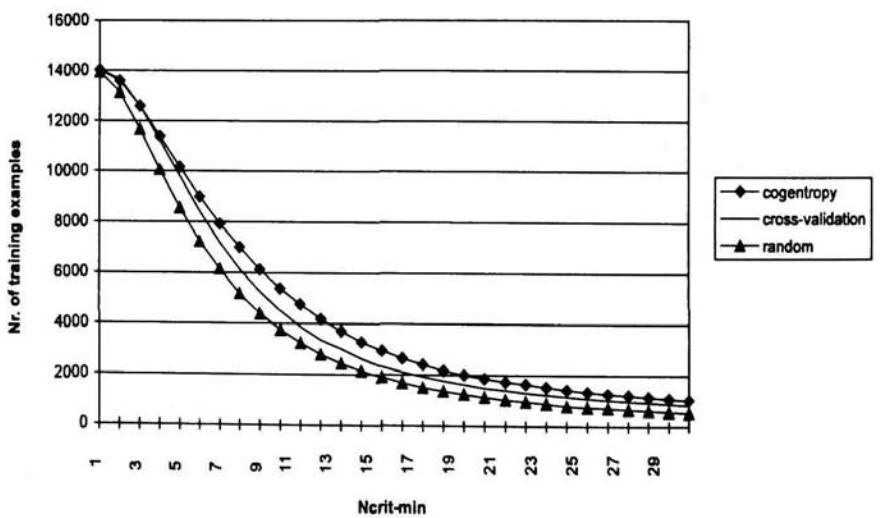


Figure 3: Cumulative distribution of n_{crit} among 14095 training examples from the NIST database.

Nr. of training examples	Nr. of test examples	Nr. of LUTs	Training method	Test Errors
14095	13521	100	cogentropy	3.5%
14095	13521	100	cossvalid.	4.0%
14095	13521	100	random	8.0%
14095	13521	200	cogentropy	3.1%
14095	13521	807	random	3.7%

Table 1: Test results obtained for the task of recognising hand-written digits

An important implication of using the cogentropy measure is illustrated in figure 3. The figure depicts the cumulative distribution of n_{crit} -values among 14095 training examples. As an example the value obtained for $n_{crit\text{-}min} = 20$ corresponds to the number of training examples having n_{crit}-values equal to or above 20. The figures show that the cogentropy training procedure gives the most robust system in the sense that more common class features in general are found.

The above results can be improved by introducing inhibitory weights into the RAM based architecture. This technique is described in an accompanying chapter[10].

4 Conclusion

We have shown that it is simple to perform a leave-one-out cross-validation test on a RAM based network. We have also introduced an information measure (denoted the cogentropy) that incorporates a generalised crossvalidating test. The information measure is used to evaluate the quality of a given combination of LUTs. The techniques were tested on the task of recognising hand-written digits. The results show that by using the information measure it is possible to obtain a considerably reduction in the number of Look Up Tables required for a given performance demand. Furthermore the use of cross-validation makes it possible to determine the number of input connection to use pr. LUT.

References

1. M. Stone, J. of Royal Stat. Soc. B, 36, 111 (1974).
2. B. Efron, J. Amer. Statist. Assoc., 78, 1, (1968).
3. L.K. Hansen and J. Larsen, Advances in Computational Mathematics, 5, 269, (1996)
4. I. Alexander and T.J. Stonham, Computers and Digital Techniques, 2, 29 (1979).
5. I. Alexander, An Introduction to Neural Computing, (Chapman and Hall, Lon-

don, 1990).
6. J. Moody 1991 in Proceedings of the first IEEE workshop on Neural Networks for Signal Processing, ed. B.H. Juang and S.Y. Kung (Piscataway, New Jersey)
7. See e.g. F.T.S. Yu, Optics and Information Theory, (John Wiley and Sons, New York, 1976).
8. T. M. Jørgensen, "Information Processing in Optical Measuring Systems," Ph.D.-thesis (Optics and Fluid Dynamics Department, Risø National Laboratory, 1992).
9. National Institute of Standards and Technology: NIST Special Data Base 3, Handwritten Segmented Characters of Binary Images, HWSC Rel. 4-1.1(1992)
10. T.M. Jørgensen, "Optimisation of RAM nets using inhibition between classes" (accompanying chapter)

A MODULAR APPROACH TO STORAGE CAPACITY

P. J. L. ADEODATO
Laboratório de Computação Inteligente, Departamento de Informática, Universidade Federal de Pernambuco, Cx. Postal 7851, 50 732 - 970, Recife - PE, Brazil

J. G. TAYLOR
Centre for Neural Networks, Department of Mathematics King's College London, The Strand WC2R 2LS, London, United Kingdom

Modularity is a valuable principle in analysing and synthesising large systems. This chapter gives an overview on how to apply such principle to the assessment of the storage capacity of RAM-based neural networks. The storage capacity of a network is a function of the storage capacity of its component neurons and subnetworks. This modular approach allows the independent treatment of storage capacity at different levels — a model for the single neuron and a model for each architecture. The technique is illustrated in the major architectures with limited storage capacity in use nowadays — general neural unit (GNU) and pyramid — and in a composition of them. The results fit well both with the existing architecture-dependent theories and with the experimental data currently available in the literature with the advantages of simplicity and flexibility for the modularity. This approach is based on collision of information during training taken as a probabilistic process.
Keywords: RAM-based neural networks, storage capacity, information collision, consistency of training sets.

1 Introduction

Theoretical issues have been capturing the attention of an increasing part of the scientific community working on Neural Networks. This chapter presents a contribution in the theory of storage capacity, a topic which has been investigated by several researchers[1–4] with different approaches to answer the following question quoted from Hertz et al.[5]

> "*How many points can we expect to put randomly in an N-dimensional space, some red and some black, and then find a hyperplane through the origin that divides the red points from the black points?*"

Here we answer the question in the context of binary input and binary output spaces with discriminant functions implementable on RAM-based neural networks.

Wong and Sherrington[1,6] analysed the storage capacity of large and dilute ($n \gg 1$ and $n \ll \log N$) general neural units (GNUs). Penny and Stonham[2]

investigated the storage capacity of pyramids. Braga[3,7] assessed the storage capacity of sparsely connected GNUs with neurons of finite fan-in (the realistic case). All those however, are architecture-dependent approaches. Recently, Adeodato and Taylor developed a modular approach for the assessment of storage capacity[4] and applied it to GNUs,[4] pyramids[8] and GNUs composed of pyramids.[9] More recently they derived a formal model for the single neuron.[10]

While single RAM-based neurons can implement any of the 2^{2^n} functions of their inputs[11] hence store all the 2^n patterns of their input space, RAM-based networks have limited functionality[12] hence limited storage capacity. Since each neuron sees only part of the network space as its input, two input patterns from different classes might have the same value on the neuron's input subspace. Therefore, consistent training sets for the network produce inconsistent ones for the neurons, which provoke an *event* referred to as *collision*[7] of information. Since collisions prevent the storage of patterns, Braga[3] accounted for their occurrence to assess the storage capacity of the GNU autoassociative memory. Later, Adeodato and Taylor[4] also accounting for them developed a technique with 3 extra features: modularity, data-dependence and flexibility. The single neuron model (module) builds up the model of any architecture, expresses the probability of collisions as a function of the training data set distribution and is valid for any RAM-based neuron model (G-RAM, pRAM, GSN). The chapter puts forward this modular approach in a modular presentation: single neuron, GNU architecture, pyramid architecture and GNU architecture with pyramids.

2 Context and Definitions

This work deals with multidimensional binary input and 1-dimensional binary output spaces. The neurons have permanent connections in the network. Input vectors (*input patterns*) address memory positions (*sites*) to store or retrieve (*train* or *recall*) the desired response (*target class*). Training is supervised. The neurons have no temporal processing power. The 2^n combinations of the n input variables connected to the n input terminals (fan-in) of the neuron address its 2^n sites. Take $\mathbf{X}$ as the discrete sample space named *input space*, which consists of the values $\{x^1, x^2, \ldots, x^{2^n}\}$. Take $\mathbf{Y}$ as the binary sample space *target class* which consists of the values y, for class-1, and $\overline{y}$, for class-2.

The training set consists of m *training examples* (loosely referred to as *patterns*), (x_i, y_i), each drawn from the stationary joint probability distribution $P\{\mathbf{X} = x, \mathbf{Y} = y\} = p(x, y)$. Take $s_m = ((x_1, y_1), \ldots, (x_m, y_m))$ to denote a training data set. Each training set is a sample (s_m) of our compound sample space ($\mathbf{S}_m = \{\mathbf{X} \times \mathbf{Y}\}^m$) of m examples. Considering that the examples are

randomly and independently drawn,

$$p((x_i, y_i), (x_j, y_j)) = p(x_i, y_i)\, p(x_j, y_j) \text{for } \; i \neq j,\; i, j \in I\!N \tag{1}$$

For the single neuron, let $\mathbf{C}$ be the binary random variable named *collision* which assumes the value c when there is at least one pair of training examples $((x_i, y_i), (x_j, y_j))$ with $x_i = x_j$ and $y_i \neq y_j$ and the value $\bar{c}$, otherwise. The value $\bar{c}$ indicates the *error-free storage*. This definition of collision is the same as that of an inconsistent training set[11] although it varies according to the neuron model or the network. For a network, the definition of collision is more complex as it may involve more than a pair of training examples. In broad terms, it takes the value c when the network is unable to store (learn) any of the mappings $f : \mathbf{X} \rightarrow \mathbf{Y}$ which fit all examples from the training set.

The term *probability of collision* measures the risk of disruption of information whereas *probability of error-free storage* measures the level of confidence in the storage process. *Storage capacity* is the maximum number of training examples (m) randomly drawn from a uniform distribution that can be stored in a neuron or network at a given risk of disruption of information (probability of collision).

For uniformly distributed examples, the probability of error-free storage ($p(\bar{c})$) is defined as the fraction of training sets correctly stored in the network and the total number of training sets of size m.

$$p(\bar{c}) = \frac{\text{number of training sets correctly stored}}{\text{total number of training sets of size } m} \tag{2}$$

Using the symbol "$|\cdot|$" to denote the cardinality of the set "$\cdot$", we can start substituting values in Equation 2. The denominator is simply $(|\mathbf{X}|\ |\mathbf{Y}|)^m$, the number of elements of the sample space $\mathbf{S}_m = \{\mathbf{X} \times \mathbf{Y}\}^m$. Omitting the functional dependence of $p(\bar{c})$ and of the numerator on $|\mathbf{X}|$ and m and considering $|\mathbf{Y}| = 2$ for the binary output space, the previous equation can be re-written as:

$$p(\bar{c}) = \frac{\text{number of training sets correctly stored}}{(2|\mathbf{X}|)^m} \tag{3}$$

This definition of $p(\bar{c})$ is valid both for single neurons (where $|\mathbf{X}| = 2^n$) and for networks (where $|\mathbf{X}| = 2^N$) in multidimensional binary input and 1-dimensional binary output spaces. Despite its generality, this equation is only useful for single neuron since the numerator is too complex to be calculated for each architecture. Hence, next section presents the model for the single neuron and the others show how to use that result to express $p(\bar{c}_{net}) = f[p(\bar{c}_{neu})]$.

3 Storage Capacity of the Single Neuron

This section presents the theoretical results for assessing the probability of error-free storage and the storage capacity of the single neuron. Emphasis is given to concepts since the formalism has been presented elsewhere.[10]

Back to Equation 3, its numerator is simply the number of consistent training data sets ($|\mathbf{S}_c|$). Therefore Equation 4 gives the probability of error-free storage ($p(\bar{c})$) for the single neuron. The calculation is still very complex. The next subsections present the exact solution for $p(\bar{c})$, an approximation ($p_a(\bar{c})$) with tight error bounds and the storage capacity of the neuron model.

$$p(\bar{c}) = \frac{|\mathbf{S}_c(m, |\mathbf{X}|)|}{(2|\mathbf{X}|)^m} \quad \text{for} \quad m \in I\!N \tag{4}$$

"*In how many ways can we configure the m patterns in a consistent fashion?*" The exact value for $|\mathbf{S}_c(m, |\mathbf{X}|)|$ is reached by a recursive method. Even eliminating recursion, the expressions remained very complex. Equations 5 and 6 are 2 non-recursive compact forms for expressing $p(\bar{c})$; one as a series along the input space dimension ($|\mathbf{X}|$) and the other along the training set size dimension (m). The full derivation is available in Adeodato and Taylor[10].

We will refer to this as the *exact model* since it was developed without any approximation. There are two drawbacks in this model however. First, Equations 5 and 6 are expensive to compute, even for $n < 5$. Second, they are not invertible for expressing the storage capacity explicitly ($m = f[p(\bar{c}), n]$).

$$p(\bar{c}) = \frac{\sum_{l=1}^{|\mathbf{X}|} (-1)^{|\mathbf{X}|-l} \binom{|\mathbf{X}|}{l} 2^l \, l^m}{(2|\mathbf{X}|)^m} \quad \text{for } m \in I\!N \tag{5}$$

$$p(\bar{c}) = \frac{\left(z \frac{\partial}{\partial z}\right)^m (z-1)^{|\mathbf{X}|}|_{z=2}}{(2|\mathbf{X}|)^m} \quad \text{for } m \in I\!N \tag{6}$$

This subsection starts from different conceptual considerations to present an approximate model for computing $p(\bar{c})$. Consistency of the training set is a function of the consistency between any two training examples, leading to $p(\bar{c}) = f[p(\bar{c}_2)]$. For a pair of examples a *collision* occurs when $x_i = x_j$ and $y_i \neq y_j$ and for a uniform distribution and a binary output space, we have:

$$p(\bar{c}_2) = \left(1 - (2|\mathbf{X}|)^{-1}\right) \quad \text{where } |\mathbf{X}| = 2^n \text{ for RAM-based} \tag{7}$$

For a training set of size m, we assume statistical independence among all $\binom{m}{2}$ possible pairs of examples. This results in a simplified model which we refer

to as the *approximate model* for the probability of error-free storage ($p_a(\bar{c})$).

$$p_a(\bar{c}) = \prod_{i=1}^{\binom{m}{2}} p(\bar{c}_2)_i = p(\bar{c}_2)^{\frac{m(m-1)}{2}} \tag{8}$$

Figure 1a shows the curves for the approximate and the exact models. The fit is striking, particularly for high fan-in. In fact, the approximate model is a first order approximation of $p(\bar{c})$. Taking $\Delta p(\bar{c}) = p(\bar{c}) - p_a(\bar{c})$ as the approximation error, its maximum value is bounded[10] by $0.125/|\mathbf{X}|$ — an exponential decrease with the fan-in. Figure 1b shows the error curves: non-negative with a single maximum which is less than 1% for $n \geq 4$. For these error bounds,

$$p_a(\bar{c}) \quad \leq \quad p(\bar{c}) \quad \leq \quad p_a(\bar{c}) + \frac{1}{8|\mathbf{X}|} \tag{9}$$

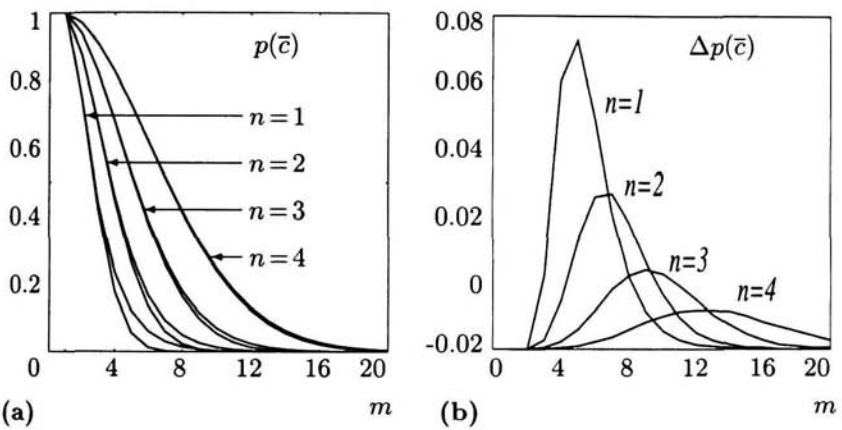


Figure 1: **(a)** Superimposition of the probability of error-free storage for the exact and the approximate models against the training set size (m) for $n = 1, 2, 3, 4$. **(b)** Error ($\Delta p(\bar{c})$) between the models.

Equations 7 and 8 are invertible hence allowing the explicit expression of the storage capacity (m) as a function of the probability of error-free storage and the fan-in ($m = f[p(\bar{c}), n]$). The bounds defined by Equations 9 can be converted into bounds for the maximum number (m) of storable training examples. This number (m) represents the *storage capacity* of the neuron for a given confidence level ($p(\bar{c})$); a standard measure in designing learning systems.

$$\frac{1}{2} + \frac{1}{2}\sqrt{1 + 8\frac{ln[p(\bar{c})]}{ln[p(\bar{c}_2)]}} \quad \leq \quad m \quad \leq \quad \frac{1}{2} + \frac{1}{2}\sqrt{1 + 8\frac{ln[p(\bar{c}) - \Delta p(\bar{c})]}{ln[p(\bar{c}_2)]}} \tag{10}$$

For a RAM-based neuron, the previous Equations (10) define the interval for the maximum number (m) of storable training examples.

$$\frac{1}{2}+\frac{1}{2}\sqrt{1+8\frac{ln[p(\bar{c})]}{ln[1-2^{-(n+1)}]}} \quad \leq \quad m \quad \leq \quad \frac{1}{2}+\frac{1}{2}\sqrt{1+8\frac{ln[p(\bar{c})-2^{-(n+3)}]}{ln[1-2^{-(n+1)}]}} \tag{11}$$

When m and n are large, the first order approximation of Equation 11 leads to the square root of an exponential function ($m \approx K\ 2^{n/2}$); much less than the exponential for the neuron trained with consistent data sets.

4 Storage Capacity of GNUs with Single Neurons

Instead of calculating the probability of error-free storage of the network from its definition (Equation 3), we use the results from the previous section together with the principle of modularity to express $p(\bar{c}_{net}) = f[p(\bar{c}_{neu})]$.

The autoassociative general neural unit (GNU)[13] does not have hidden layers and training in each neuron occurs in the same context as in Section 3. As the GNU has N output neurons, a collision in any of them represents a collision to the whole network. Thus error-free storage only occurs in the GNU when it does in all its neurons simultaneously. Considering the statistical independence among the neurons (patterns randomly drawn and neuron connections randomly set), the probability of error-free storage of the GNU ($p(\bar{c}_{gnu})$) is simply the product of the probabilities for its neurons ($p(\bar{c}_{neu})$):

$$p(\bar{c}_{gnu}) = p(\bar{c}_{neu})^N \tag{12}$$

At This point we need a theoretical result: *A neuron with input to itself* (self-feedback) *never produces collisions in an autoassociative GNU*.[3] Knowing the probability of self-feedback ($p(f)$) and combining it with the results from the Section 3 ($p(\bar{c})$), we obtain $p(\bar{c}_{neu})$ for substituting in Equation 12.
$p(\bar{c}_{neu}) = p(f) + p(\bar{f})\ p(\bar{c})$.

Here we present the model for the GNU in the same conditions as experimentally tested and modelled by Braga[3] for comparison of results. The connections are randomly sampled from the space of dimension N without replacement determining a probability of self-feedback of $p(f) = n/N$. Therefore the probability of error-free storage in a GNU ($p(\bar{c}_{gnu})$) of dimension N is:

$$p(\bar{c}_{gnu}) = \left(\frac{n}{N} + \left(1-\frac{n}{N}\right) p(\bar{c})\right)^N \tag{13}$$

The inversion of Equation 13 gives the explicit expression for the storage capacity of the GNU (Equation 14) which is substituted in the storage capacity

equation for the neuron model in Section 3.

$$p(\bar{c}) = \frac{(p(\bar{c}_{gnu}))^{1/N} - \frac{n}{N}}{1 - \frac{n}{N}} \quad \text{for } n \neq N \tag{14}$$

Figure 2 shows the probability of collision calculated with $p_a(\bar{c})$ under the same angle of vision as Braga's[3] experimental results. The model seems to fit strikingly with the data but no numerical comparison has been made so far.

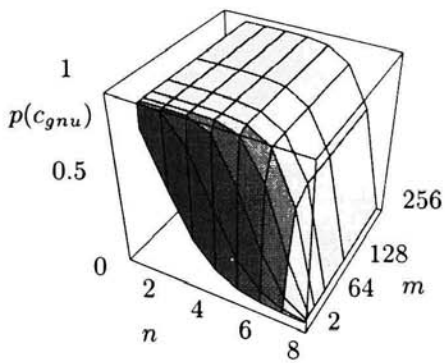


Figure 2: Probability of collision in an 8-dimensional GNU for varying fan-in and number of stored patterns. Check against Braga's actual data.[3]

In fact, Equations 13 and 14 can be used to define the interval where the actual values lie by just using Equations 9 and 10, respectively. This straightforward composition for the bounds is valid because the GNU model is exact. That is; does not add any other error just transforms the existing one.

Figure 3 presents $\log_2$ of the storage capacity against the fan-in (n) for a confidence level of 0.5 ($p(\bar{c}) = 0.5$) for several models (including Braga's[7] Equation 13 and Wong and Sherrington's[6] equation in paragraph 2 page 419). Figure **3a** shows that Braga's model[7] underestimates the storage capacity of the GNU and that his estimate is outside the error margins. Figure **3b** shows that Adeodato and Taylor's old models[4] overestimate the storage capacity of the GNU in agreement with the overestimation of their old neuron models. It also shows that Wong and Sherrington's model[6] overestimates the storage capacity of the GNU. Not surprisingly, Figure **3b** shows Braga's model[7] and ours nearly superimposed. This is expected because when m and n are large, both models can be approximated by the square root of an exponential function ($m \approx K\, 2^{n/2}$).

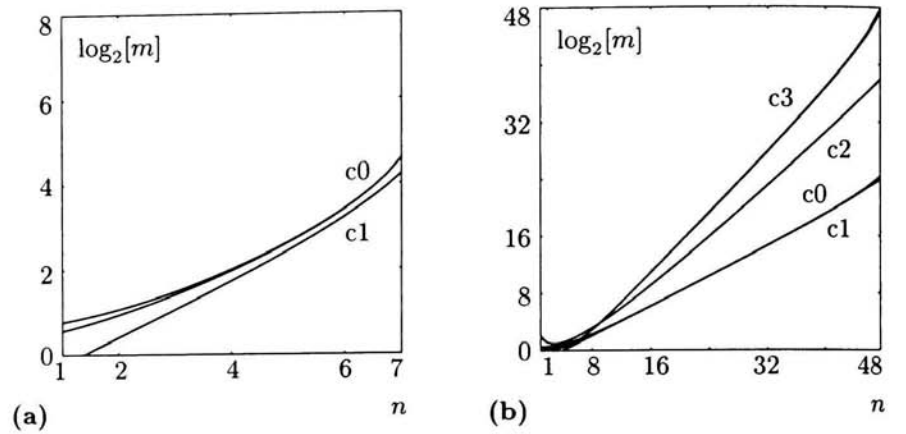


Figure 3: $\log_2$[storage capacity]× fan-in (n) for a confidence level of 50% ($p(\bar{c}) = 0.5$) for several models. **(a)** 8-dimensional GNU with our model's interval (c0) and Braga's model (c1). **(b)** 50-dimensional GNU with our model's interval (c0), Braga's model[7] (c1), Wong and Sherrington's model[5] (c2) and Adeodato and Taylor's old models[4] (c3).

5 Storage Capacity of Pyramids

For pyramids, the application of the principle of modularity to express $p(\bar{c}_{net}) = f[p(\bar{c}_{neu})]$ is not straightforward. The hidden layers learn internal representations over which we have no control. Let us define pyramid as a tree of RAM-based neurons arranged in layers.[13] Its N input terminals are connected to the N input features in a 1-to-1 fashion, sampling the space in $\frac{N}{n}$ mutually exclusive sets of n features.

The crucial point to apply the modular approach to pyramids is how to overcome the lack of information about the neurons in the hidden layers. The principle of memory occupation that maximises storage is the key: *A collision is only propagated by a neuron to its successor (the one which receives its output) if* **all** *its antecessors (the neurons which provide its inputs) have suffered a collision.* This is because the non-colliding neuron's output divides the addressable sites of its successor in two disjoint sets, each set associated with a different class (target).

We assume that the patterns are randomly drawn from a uniform distribution. This assumption together with the sampling of mutually exclusive sets of features and the tree structure of the pyramid assure that collisions among the antecessor neurons are statistically independent events. Therefore the probability of collision — $p(C_l)$ — in a neuron of fan-in n_l of layer l is simply the

product of the probabilities of collision among its l preceding neurons. $p(c_l)$ is just the neuron model ($p(c)$ from Section 3). Hence, the building block for assessing storage capacity in pyramids is[8]:

$$\begin{aligned} p(C_1) &= p(c) \\ p(C_l) &= p(c_{(l-1)})^{n_l} \qquad \text{for} \quad l = 2, 3, \ldots L \end{aligned} \tag{15}$$

The probability of collision in the pyramid ($p(c_{pyr})$) is the effect of neuron collisions propagated from the input to the output layer (L). For uniform pyramids (equal neurons), it simplifies to:

$$p(c_{pyr}) = p(c)^{(n(L-1)+1)} \tag{16}$$

Figure 4 plots Equation 16 for the same pyramids used by Penny and Stonham[2] keeping the same aspect ratio of their plots. We use the exact model for $p(\bar{c})$. It produces a smooth curve which fits with the data qualitatively better. The underestimation of $p(\bar{c}_{pyr})$ was expected because our assumption of sampling with replacement differs from their experimental set-up with the draws without replacement. This difference however disappears when pyramids of more layers are considered. The number of possible input patterns ($|\mathbf{X}|$) increases exponentially ($|\mathbf{X}| = 2^{(nL)}$) while the storage capacity remains nearly constant ($m \approx 2^n$). Drawing only 2^n examples from such a huge universe is approximately the same for drawings with or without replacement. Therefore, the bigger the depth of the pyramid, the higher the precision of our model.

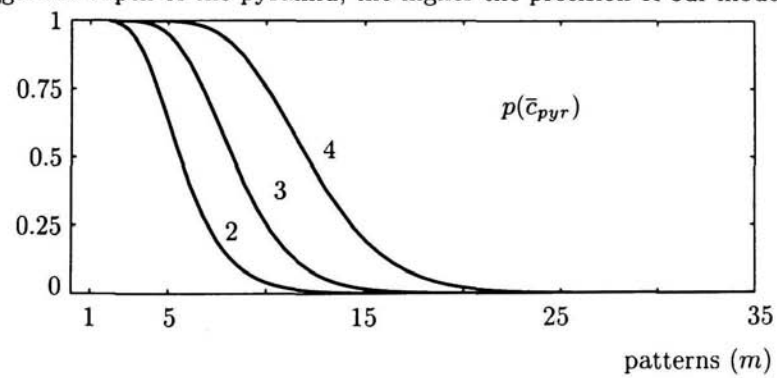


Figure 4: Probability of error-free storage × number of patterns in 2-layer pyramids with neurons of fan-in 2, 3, 4 (exact model). Check against Penny and Stonham's actual data.[2]

There is no additional difficulty to apply this technique to large networks. Figure 5a for 256-pyramids shows a sharp transition of the curves and how they scale with the fan-in. Applied to 16-pyramids of different fan-ins per layer, this

approach produced qualitatively the same result as Al-alawi and Stonham's.[12]

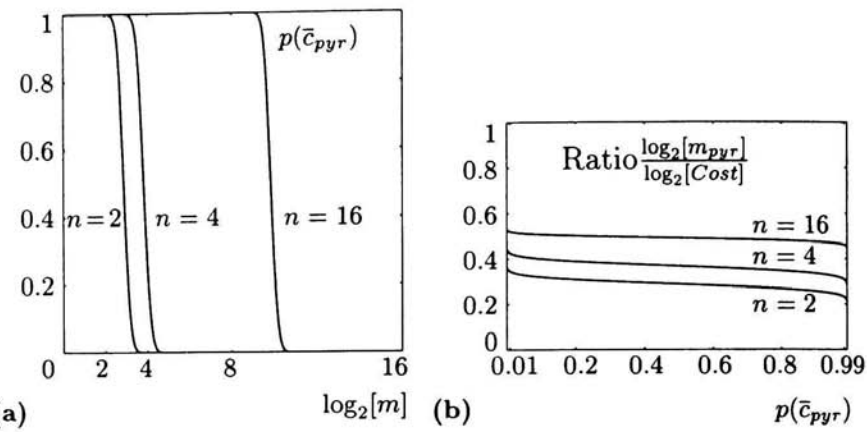


Figure 5: **(a)** Probability of error-free storage × $\log_2$[number of patterns] in 256-pyramids. **(b)** Ratio of $\log_2$[storage capacity] over $\log_2$[cost] as a function of the probability of error-free storage in 256-pyramids. Each pyramid has equal neurons, with fan-ins of 2, 4 and 16.

Inverting Equation 16 we get the explicit Equation (17) for storage capacity of the pyramid as a function of the desired $p(c_{pyr})$ which is then substituted in the storage capacity for the neuron (m, in Section 3). With cost measured as the total amount of memory in the network[8], Figure 5**b** presents the ratio of $\log_2$[storage capacity] and $\log_2$[cost] as a function of $p(\bar{c}_{pyr})$. The reduction in storage capacity is bigger than the reduction in cost when we reduce the fan-in of the neurons. This "extra" reduction is caused by the multiplicity of representation of the functions, explained by Al-alawi and Stonham.[12] This is another result coherent with previous architecture-specific techniques.

$$p(c) = 2^{\left(\frac{\log_2 p(c_{pyr})}{n(L-1)+1}\right)} \tag{17}$$

6 Storage Capacity of GNUs with Pyramids

The assessment of the storage capacity of GNUs with pyramidal neurons is the natural step after having developed the models both for the GNU and the pyramid due to the modularity of our approach. The approach is the same as that applied to the GNU with single neurons, only with slightly different considerations. As pyramids receive inputs from the whole space, there is always self-feedback. However, by not having full functionality, pyramids have a

chance of provoking collisions in a pyramid-GNU, even with self-connections. So error-free storage only occurs in the pyramid-GNU when it does occur in all its pyramids simultaneously. As for the GNU with single neurons, we obtain the probability of error-free storage for the pyramid-GNU ($p(\bar{c}_{gnupyr})$, Equation 18 from the probability of collision of the pyramids ($p(c_{pyr})$). Inverting Equation 18) we get the storage capacity for the pyramid-GNU which is substituted in Equation 17 for the pyramid.

$$p(\bar{c}_{gnupyr}) = (p(\bar{c}_{pyr}))^N \tag{18}$$

$$p(\bar{c}_{pyr}) = (p(\bar{c}_{gnupyr}))^{\frac{1}{N}} \tag{19}$$

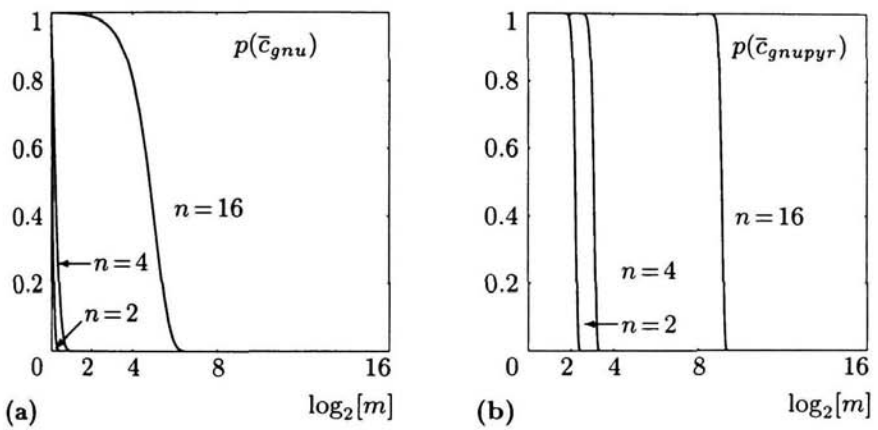


Figure 6: Probability of error-free storage in a 256-GNU as a function of $\log_2$[number of patterns] for different fan-ins ($n = 2, 4, 16$). **(a)** single neurons, **(b)** pyramids.

Figure 6 shows the advantage of having pyramids instead of single neurons in a 256-GNU. The gain is impressive, particularly for high confidence levels. An additional gain is the increase in the storage capacity/cost ratio.[9]

7 Conclusions

This chapter has presented a modular approach to assess storage capacity of RAM-based neural networks for patterns randomly drawn from a uniform distribution. The consistency of the results at the architecture level for different single neuron models — the originally proposed model[4] and the precise one[10] — confirms the validity of the modularity principle.

The modularity of the approach allowed the straightforward assessment of the storage capacity of the GNU with pyramids. It proved there is a huge increase in storage and storage/cost ratio when compared to the GNU with single neurons. These are the first results on the architecture proposed 7 years ago by Aleksander[13] and are relevant for the practical application of such architecture as autoassociative memory using any RAM-based neuron.

Simplicity and exponentially increasing precision with the fan-in for the approximate model of the neuron raised the quality in the assessment of storage capacity by lowering computational costs and giving high precision at the architecture level. The model for the autoassociator GNU is also very precise since the actual data distribution coincides with that assumed for the neuron allowing the definition of the storage capacity with error margins for the GNU.

For the pyramids, the technique has avoided the hard learning problem by defining conditions for optimal storage in hidden layers. The results are the best so far and fit well with all data currently available. Nevertheless, further experimental testing has to be done for large input spaces to check the claimed increase in the precision of the model with the number of layers as the actual and the assumed training data sets tend to follow the same distribution.

References

1. K. Y. M. Wong and D. Sherrington, Europhysics Letters **7**, 197 (1988).
2. W. Penny and T. J. Stonham, Electronics Letters **29**, 1340 (1993).
3. A. P. Braga, Electronics Letters **30**, 55 (1994).
4. P. J. L. Adeodato and J. G. Taylor, in *Proc. of WNNW'95*, ed. D. L. Bisset, Canterbury, UK, 1995.
5. J. Hertz, A. Krogh and R. G. Palmer, *Introduction to the theory of Neural Computation*, 90–113, (Addison Wesley Publishing Co., 1991).
6. K. Y. M. Wong and D. Sherrington, J. Phys. **22**, 2233 (1989).
7. A. P. Braga, Tech. Rep. APB-1/92, Imperial College, Dept. Elect. Electron. Eng., London, UK, 1992.
8. P. J. L. Adeodato and J. G. Taylor, Neur. Net. World **6**, 241 (1996).
9. P. J. L. Adeodato and J. G. Taylor, in *Proc. of ICANN'96*, eds. C. von der Marlsburg and W. von Seelen, Bochum, Germany, 1996.
10. P. J. L. Adeodato and J. G. Taylor, in *Proc. of SBRN'96*, ed. T. B. Ludermir, Recife, Brazil, 1996 (available at http://www.di.ufpe.br/~ pjla).
11. M. Anthony and N. Biggs, in *Mathematical Approaches to Neural Networks*, ed. J. G. Taylor, (Elsevier Science Publishers, 1993).
12. R. Al-Alawi and T. J. Stonham, Electronics Letters **25**, 657 (1989).
13. I. Aleksander, IEE Comp. & Cont. Eng. J. **1**, 259 (1990).

GOOD-TURING ESTIMATION FOR THE FREQUENTIST N-TUPLE CLASSIFIER

M. MORCINIEC[†], R. ROHWER[§],
Neural Computing Research Group,
Aston University, Birmingham, B4 7ET, UK

We present results concerning the application of the Good-Turing (GT) estimation method to the frequentist n-tuple system. We show that the Good-Turing method can, to a certain extent, rectify the Zero Frequency Problem by providing, within a formal framework, improved estimates of small tallies. We also show that it leads to better tuple system performance than Maximum Likelihood Estimation (MLE). However, preliminary experimental results suggest that replacing zero tallies with an arbitrary constant close to zero before MLE yields better performances than those of a GT system.

1 Introduction

The frequentist n-tuple system can be obtained from the original, binary system by setting the tally truncation threshold θ to ∞ instead of the more usual 1. This allows one to use full tallies to estimate low-order conditional feature densities and apply a Bayesian framework to the classification problem.[6]

Given $p(c|\alpha)$, the probability of class c conditioned on feature vector α (the set of all memory locations addressed by an unknown pattern), optimal classification results can be obtained by assigning the unknown pattern to the most probable class. Because estimates of conditional feature densities arise naturally in the n-tuple system, Bayes' rule is applied to obtain class probabilities. The likelihood and evidence for the full feature vector are impossible to compute directly, but these can be estimated from low order densities using independence assumptions. The most common approach[11] assumes that $p(\alpha_i|c)$ as well as $p(\alpha_i)$ are independent[‡], where α_i is the address of the pattern in n-tuple i. The conditional class density can then be approximated by

$$p(c|\alpha) \approx p(c)\prod_i \frac{p(\alpha_i|c)}{p(\alpha_i)} \tag{1}$$

[†] Current address: Hewlett-Packard Labs, Filton Rd., Stoke Gifford, Bristol BS12 6QZ, UK

[§] Current address: Prediction Company, 320 Aztec St., Suite B Santa Fe, NM, 87510, USA, email: rr@predict.com

[‡] It often goes unnoticed that it turns out to be highly restrictive to demand both of these conditions together, a difficulty we presume to be dwarfed by the inaccuracy of each assumption individually.

However implausible this assumption may appear, there have been reports of reasonable results obtained with this method [3,2]. The major advantage of frequentist systems is that they do not suffer from saturation. This makes them superior for small n-tuple sizes n, but the advantage tends to disappear as n is increased, due to worsening probability estimates based on diminishing tallies in each of the increasingly numerous memory locations [12]. It would be desirable to modify the frequentist system in such a way as to retain its robustness for any tuple size n.

2 Weakness of the Maximum Likelihood Estimate (MLE)

The maximum likelihood estimate (MLE) has been routinely [3,11,13] applied for the frequentist n-tuple system. In this approach, estimate $\hat{p}$ of the true probability p of an event is approximated as the ratio of the event's tally r to the sample size N; $\hat{p} = \frac{r}{N}$. Under the assumption that each tally value is binomially distributed (with unknown probability p that the feature is present in a pattern of class c and $1 - p$ that it is not) the ratio $\frac{r}{N}$ is the maximum likelihood estimate of p. The uncertainty of the tally can be defined as its standard deviation, which can be estimated as

$$\delta r = \sqrt{Np(1-p)} \approx \sqrt{N\hat{p}(1-p)} = \sqrt{r(1-p)}. \tag{2}$$

In an n-tuple with n inputs, p is one of $2^n - 1$ other multinomial parameters which sum to 1. Therefore, p is typically much less than 1, so

$$\delta r \approx \sqrt{r}. \tag{3}$$

Equation 3 shows that the accuracy of MLE is limited for the events with small tallies. The relative tally uncertainty $\frac{\delta r}{r}$, grows with diminishing tallies and becomes undefined for zero tally.

It should be noted that the fact that a tally $r = 0$ for some event doesn't imply that the probability of the event is also zero. It merely states that the event has not taken place in a finite sample of size N. This problem is known in the literature [14] as the "Zero Frequency Problem". Various unprincipled, *ad hoc* techniques exist which try to rectify it. The most common one is to add an arbitrary small constant to each zero tally. However, the choice of a particular constant is difficult to justify formally. We make some experimental observations concerning this Maximum Likelihood system with zero tally correction (MLZ) in section 5.

3 Good-Turing Estimate (GTE)

An alternative method of density estimation has been originally proposed by Turing and researched in detail by Good [4] in the context of species frequencies in a mixed population. It has also been applied in linguistics for n-gram probability estimation [5] and statistical text compression [14]. The advantage of GTE over MLE is improvement of the accuracy of the probability estimates derived from non-zero tallies. Moreover, an estimate for objects not present in the sample can also be provided.

Suppose we draw a random sample of size N from the population of objects. We record n_r, the number of distinct objects that were represented exactly r times in the sample, so that

$$N = \sum_{r=1}^{\infty} r n_r. \tag{4}$$

Let $\hat{p}_r^{GT}$ denote the Good–Turing estimate of the population probability of an arbitrary object that occured r times in the sample. This entails the assumption that all events which occured r times have the same probability p_r. The Good–Turing theorem states that the expected value of p_r for an event with tally r in one particular sample is r^{GT}/N where the smoothed tally r^{GT} can be approximated as

$$r^{GT} \approx (r+1)\frac{n_{r+1}}{n_r} \qquad r \geq 0. \tag{5}$$

Various derivations [7] of this theorem exist. The values of n_r are most accurate for small values of r and become increasingly noisy for larger tallies. In this respect GTE complements MLE which, as mentioned earlier, becomes less precise for smaller tallies.

4 Smoothing GTEs

The major problem with the Good–Turing theorem is that the distribution $\{n_0, n_1, n_2, \ldots\}$ tends to be sparse and requires smoothing. Moreover, for large values of r there are "gaps" in the distribution of n_r. This suggests that we should average a non-zero n_r value with the zero n_r values surrounding it. We use the transform proposed by Church and Gale [5]

$$z_r = \frac{2n_r}{t-q} \tag{6}$$

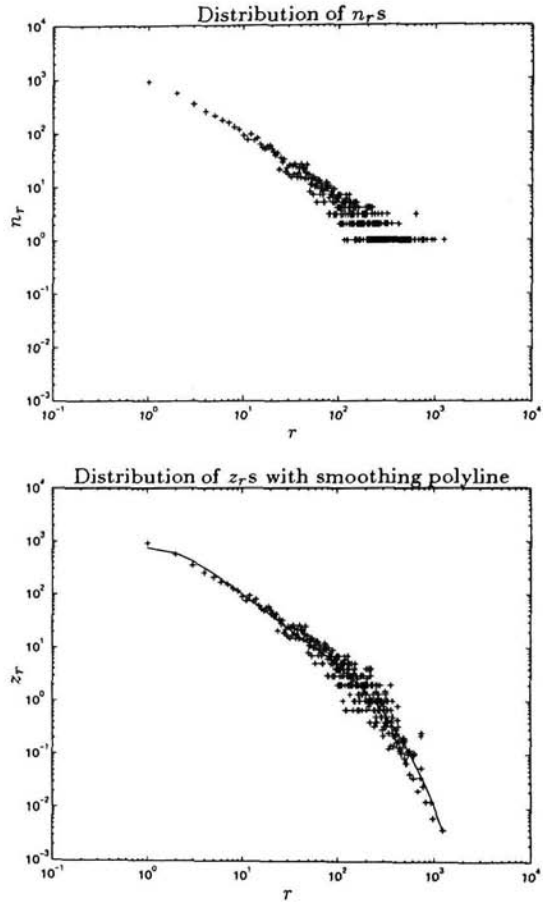


Figure 1: A) Original distribution of frequencies of tallies in class 0 of "tsetse" database. B) distribution after averaging transform has been applied. The solid line denotes the polynomial curve fitted.

where t, r, q are the successive indices of non-zero n_r. Averaging occurs for larger values of r only, because if there are no "gaps" the transformation has no effect.

After averaging we still have to smooth the z_r. This is accomplished by fitting a log polynomial onto the data. Unlike Church and Gale who used polynomial of order one (a straight line) we found that polynomials of higher

orders are required to obtain a satisfactory fit to the data. Consequently, we smoothed tally frequency distributions z_r with polynomials of order 4, giving a new smoothed tally r^{SGT},

$$r^{SGT} = (r+1)e^{\sum_{i=1}^{n} a_i \ln^i \frac{r+1}{r}} \qquad r \geq 1 \tag{7}$$

with parameters $a_1, a_2, \ldots, a_4$ determined from the data. Figure 1 shows the original n_r, and averaged z_r distributions with the fitted polynomial curve.

The smoothed Good–Turing estimate (SGTE) may be quite different from the original Good–Turing estimate (GTE) for small values of r. We would therefore prefer to use GTE for small r and then switch to SGTE and keep on using this estimate for the remaining tally values. The new, composite smoothed tally r^* is equal to r^{GT} if $|r^{SGT} - r^{GT}| > 1.65 \times \sigma(r^{GT})$. When the difference becomes insignificant we use SGTE for the remaining tallies. Gale gives the approximation of the variance of r^{GT} as

$$\sigma^2(r^{GT}) \approx (r+1)^2 \frac{n_{r+1}}{n_r^2}(1 + \frac{n_{r+1}}{n_r}) \tag{8}$$

The probability estimates computed using the corrected tallies have to be normalised because two different methods (GTE and SGTE) of estimation are employed. We compute the probability $\hat{p}_r^{norm}$ for the tally r using unnormalised probabilities $\hat{p}_r^* = \frac{r^*}{N}$ as

$$\hat{p}_r^{norm} = \begin{cases} (1 - \frac{n_1}{N}) \frac{\hat{p}_r^*}{\sum_{r' \geq 1} n_{r'} \hat{p}_{r'}^*}, & \text{when } n_0 \neq 0 \text{ and } r \geq 1 \\ \frac{n_1}{n_0 N}, & \text{when } n_0 \neq 0 \text{ and } r = 0 \\ \frac{\hat{p}_r^*}{\sum_{r'} n_{r'} \hat{p}_{r'}^*}, & \text{when } n_0 = 0 \text{ and } r \geq 1 \end{cases} \tag{9}$$

5 Application of GTE for the Frequentist n-tuple System

We used several real-world datasets which have been used in the European Community StatLog project [10]. The attributes of the data are in most cases real-valued and pre-processing techniques have been applied [1,9,8], providing binary input for the classifier.

In order to obtain probabilities $p(\alpha_i|c)$ normalised within a tuple node one would have to apply Good–Turing estimation for each tuple in each class c separately. This is hardly possible because the distribution n_r is very sparse, especially for small tuple sizes n. Therefore, the estimation has been carried

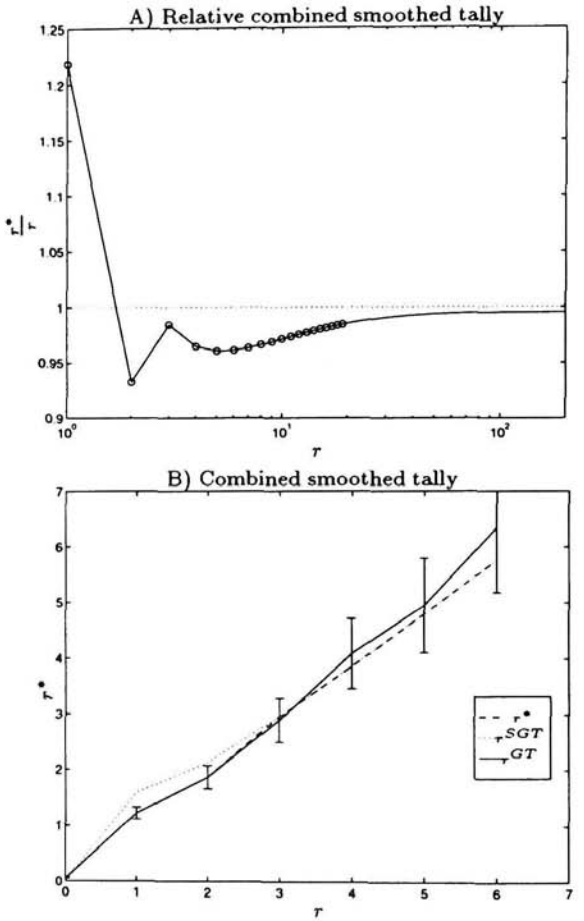


Figure 2: A) Relative adjusted tally r^* for class 0 of the "tsetse" dataset. B) Illustration of the smoothed tally r^* combined from tallies r^{GT} and r^{SGT}. The error-bars on r^{GT} are $1.65 \times \sigma(r^{GT})$. The switch from GTE to SGTE takes place at $r = 3$.

out collectively for all T tuples within a class c, i.e., for the population of $T2^n$ features. Consequently, the probabilities $p(\alpha_i|c)$ are normalised within each discriminator c and each zero tally is smoothed by the same amount regardless of the tuple which generated it.

Figure 2A shows the relative composite smoothed tally r^*/r computed for the first discriminator of the n-tuple system trained on the "tsetse" dataset [10].

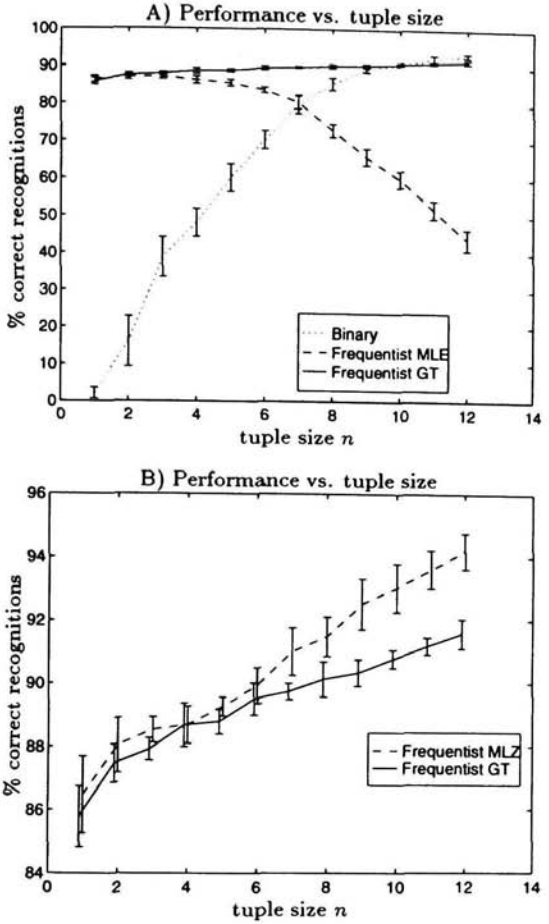


Figure 3: A) Performance of binary and frequentist systems with 100 tuples on the "tsetse.tst". The test set comprises of 1499 patterns. Systems were trained using 3500 training samples. The error-bars are of size one standard deviation computed for 10 random tuple mappings. B) Performance of frequentist Good–Turing system compared with MLZ system using zero tally correction $\epsilon = 10^{-150}$.

The construction of the combined smoothed tally r^* is given on figure 2B. We observe that for the first three tallies GTE was chosen whereas SGT was used for the remaining tallies. The adjusted zero tally was $r_0^* = 0.0452$.

We measured the performance of the binary, frequentist ML and GT n-

tuple systems against each other. The benchmarking studies were carried out for several STATLOG databases [10]. Figure 3A shows a representative plot for a run on the "tsetse" database. Both frequentist systems perform better than the binary version for small values of n, because they do not suffer from the saturation effect. Unlike the frequentist system with MLE, the GT version retains the performance with increasing n. However, it eventually becomes inferior to binary system. It seems that for n large enough any technique other than zero tally counting (which is equivalent to setting the tally truncation threshold θ to one) is less effective.

We also compared the performance of the GT system to that of MLZ which is technically an ML system with zero tallies substituted by arbitrarily chosen constant ϵ. Preliminary experimental results plotted on Figure 3B suggest that if ϵ is small enough then MLZ will outperform GT system, especially for large n. This can be explained by observing that MLZ with $\epsilon \rightarrow 0$ will make exactly the same classification decision as the binary system, except for the patterns that are tied (have the same score) in the binary version. For large n, the saturation is very low, as is the probability of a tie. Consequently, the performance of MLZ must be equal to the performance of a binary system within a margin $\pm\frac{D_{tied}}{D}$ where D is test set size and D^{tied} number of tied patterns.

6 Conclusions

We have demonstrated that a major weakness of the frequentist tuple system using MLE is inadequate probability estimation for small tallies. A principled approach to tally smoothing using Good-Turing formula leads to an improved system performance for larger values of n. However, experiments suggest that replacing zero tallies with a small constant and using a maximum likelihood estimate yields even better results.

Acknowledgements

The authors are grateful to Trevor Booth of the Australian CSIRO Division of Forestry for permission to report results on the Tsetse data set, and William Gale of AT&T Bell Laboratories for private communication.

References

1. N.M. Allinson and A. Kołcz. Application of the cmac input encoding scheme in the n-tuple approximation network. *IEE Proceedings on Com-*

put. Digit. Tech., 141(3):177–183, 1994.

2. Ardeshir Badr. N-tuple classifier for ecg signals. In N. Allinson, editor, *Proceedings of the Weightless Neural Network Workshop '93, Computing with Logical Neurons*, pages 29 – 32, University of York, 1993.
3. W.W. Bledsoe and C.L. Bisson. Improved memory matrices for the n-tuple recognition method. *IRE Joint Computer Conference*, 11:414–415, 1962.
4. I.J. Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3):237–263, 1953.
5. W.A. Gale K.W. Church. A comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of english bigrams. *Computer Speech and Language*, 5:55–64, 1991.
6. C.N. Liu. A programmed algorithm for designing multifont character recognition logics. *IEEE Transactions on Electronic Computers*, 139(2):586–593, 1964.
7. A. Nadás. On Turing's formula for word probabilities. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 33(6):1414–1416, 1985.
8. R. Rohwer and M. Morciniec. Benchmarking the n-tuple classifier with Statlog datasets. In D. Bisset, editor, *Proceedings of the Weightless Neural Network Workshop 1995, Computing with Logical Neurons*, pages 29 – 34, University of Kent, 1995.
9. R. Rohwer and M. Morciniec. The theoretical and experimental status of the n-tuple classifier. Technical Report NCGR/4347, Aston University Neural Computing Research Group, Aston Triangle Brimingham B4 7ET UK, 1995.
10. D. Michie D.J. Spiegelhalter and C.C. Taylor, editors. *Machine Learning, Neural and Statistical Classification*. Prentice–Hall, 1994.
11. M.J. Sixsmith G.D. Tattersall and J.M. Rollett. Speech recognition using n-tuple techniques. *British Telecom Technology Journal*, 8(2):50–60, 1990.
12. J.R. Ullmann. Experiments with the n-tuple method of pattern recognition. *IEEE Transactions on Computers*, 18(12):1135–1137, 1969.
13. J.R. Ullmann and P.A. Kidd. Recognition experiments with typed numerals from envelopes in the mail. *Pattern Recognition*, 1:273–289, 1969.
14. I.H. Witten and T.C. Bell. The zero–frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4):1085–1094, 1991.

PARTIALLY PRE-CALCULATED WEIGHTS FOR BACKPROPAGATION TRAINING OF RAM-BASED SIGMA-PI NETS

R NEVILLE
Division of Elec. Eng., School of Engineering, University of Hertfordshire, Hatfield campus, College Lane, Hatfield, Herts. AL10 9AB, UK

The chapter outlines recent research that enables one to train digital "Higher Order" sigma-pi artificial neural networks using pre-calculated constrained look-up tables of Backpropagation delta changes. By utilising these digital units that have sets of quantised sitevalues (i.e. weights) one may also quantise the sigmoidal activation-output function and then the output function may also be pre-calculated. The research presented shows that by utilising weights quantised to 128 levels these units can achieve accuracy's of better than one percent for target output functions in the range $Y \in [0,1]$. This is equivalent to an average Mean Square Error (MSE) over all training vectors of 0.0001 or an error modulus of 0.01. The sigma-pi are RAM based and as such are hardware realisable units which may be implemented in Microelectronic technology. The article present a development of a sigma-pi node which enables one to provide high accuracy outputs utilising the cubic node's methodology of storing quantised weights (site-values) in locations that are stored in RAM-based units. The networks presented are trained with the Backpropagation training regime that may be implemented on- line in hardware. One of the novelties of this work is that it shows how one may utilise the bounded quantised site-values (weights) of sigma-pi nodes to enable training of these Neurocomputing systems to be relatively simple and very fast.

1 Introduction

Most of the current work with Artificial Neural Networks uses nodes whose activation is defined via a linear weighted sum of the inputs, and whose output is then obtained by passing this through a sigmoidal squashing function. The linearity required immediately places restriction on the node functionality and, although it is possible to implement any continuous function using two layers of such nodes, the resources required in terms of hardware and time may be prohibitive. Further, biological nets make use of non-linear activation components in the form of axo-axonic synapses performing presynaptic inhibition. The simplest way of modelling such synapses and introducing increased node complexity is to use multi-linear activation, that is the nodes' activation is now the sum of product terms or is in 'Sigma-pi' form[1]. Units of this type are also designated 'higher order' in that they contain polynomial terms of order greater than one (linear). Our Sigma-pi units are 'higher order' nodes and as such make use of non-linear activation components. Recent research by Lenze[2], has shown how to make

Sigma-pi Neural Networks perform perfectly on regular training sets.

A quite different approach to that of the majority of researchers who utilise semi-linear nodal models for neural network construction has been taken by Aleksander & Stonham[3], Wilkie[4], Kan[5], Zhang et al.[6] and Austin[7]. This has been primarily implementation driven and has, at its point of departure, the observation that any Boolean function may be implemented as a look-up table in Random Access Memory (RAM).

In addition, the work on probabilistic units of Aleksander[8], Gorse and Taylor[9] and Gurney[10] has laid down the basic theory for Feedforward structures. Gurney's work initiated the work of Neville[11] to evaluate hardware topologies for Feedforward neural networks which combine network and training hardware on a single chip.

The research of Gurney[10] has developed several learning rules for Sigma-pi units, these include Associative Reward-Penalty, Back-Propagation of errors and System Identification. The research into Back-Propagation of errors for Sigma-pi units of Gurney [10] was derived from the standard semi-linear Back-Propagation of Werbos[12] and then Rumelhart, Hinton and Williams[1]. This is based on the back propagation of a delta term which is derived from the previous node's output error times the weight of the connection associated with it. Gurney's breakthrough was the interpretation of these connection weights as probabilities which he derived from bit-streams, to provide what he terms 'dynamic weights'. However this research does not use Gurney's bit-stream methodology to derive the weights on the connections of the nodes as bit-streams may have associated problems. Instead the instantaneous dynamic weight is utilised, as it can be derived for the present forward pass and does not have to be assimilated over multiple Feedforward operations as was previously the case with Gurney's dynamic weights derived from bit-streams. By utilising these dynamic weights the Back-Propagation paradigm of the semi-linear units can then be directly interpreted into a Sigma-pi notation of digital Sigma-pi (probabilistic) logic nodes. The benefits of utilising the Sigma-pi nodal model with this regime is that it uses stochastic state space search methods and the nodal model itself has an inherent ability to introduce noise while it is being trained.

Recent research has investigated the suitability of a Massively Parallel Processor (MPP), the Associative String Processor (ASP), for Neurocomputing. This research has lead to several papers, one of which was entitled 'Evaluation of training Sigma-pi Networks on a Massively Parallel Processor'[13]. The research mapped these neuron models to the ASP utilising low level ASP code. The research utilised a methodology of pre-calculated constrained look-up tables to store the Sigma-pi's output function $f(S_\mu)$, the delta changes ΔS_μ for the Associative Reward-Penalty training regime and the output error per visible node $e_o(S_\mu)$. This means that computationally intensive mathematical functions, such as the sigmoidal function, are pre-calculated prior to the run time of the system. Previous work using the ASP with Error Back-propagation training has had to utilise an approximated sigmoid using piece-wise linear func-

tions, Grozinger[14].

2 Rationale

The recent evolution of real-valued input RAM-based Sigma-pi networks by Gurney[10] has enabled hardware realisable systems to be built that utilise cheap RAM-based hardware. To map real-valued functions with these units and acquire high accuracy outputs one needs to address the following questions:

- how to input high accuracy real-values to the net;
- how the high accuracy values are transmitted through the system;
- how the machine instantaneously represents the values; and
- how real functions are mapped onto these systems.

The following paragraphs relate how this may be done utilising what is known as a Real- valued Activation Time Integration Sigma-pi unit.

3 Introduction to The Real-Valued Activation S-Model sigma-pi unit

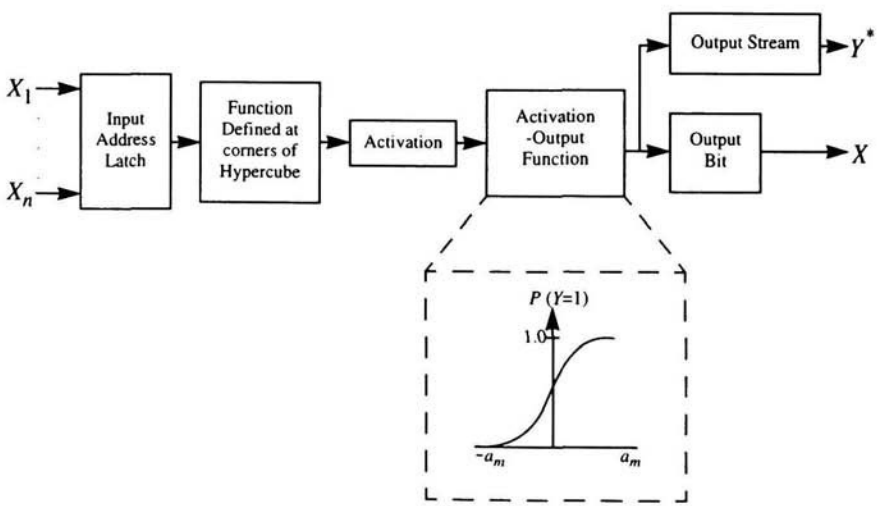


Figure 1. The Real-Valued Activation Time Integration sigma-pi unit.

The sigma-pi model[15,16,17], which in the case of this research take the form of a Stochastic-Model (S-Model) as the site-values are interpreted as probabilities. The new model that has been developed is termed a Real-Valued Activation Time Integration

Node (RVA-TIN), and contains a hypercube of sites which are averaged over time to provide a Real-Valued Activation which is then interpolated through an Activation-output function, the RVA-TIN is shown in Figure 1.

The site-values, $S_{\mu_1, \ldots, \mu_n}$ are stored in an array (n-tuple), which may be stored sequentially in locations in RAM. The input x_i may also be interpreted as the probability of a "1" appearing at the ith input to the node. The output y is defined by a probabilistic process which is presented in the following paragraphs. A Time Integration Node stores site-values in a hypercube, which is addressed by an i bit input vector. The input vector addresses a site μ which contains a site-value S_μ which stores a value $S_\mu \in \{-S_m, \ldots, S_m\}$ which is interpreted as a quantised number for reasons of hardware implementation, but it may also be a real number. The site-value is then used to estimate the activation which is defined in a similar manner to Barto. et al.[18]. The activations, $\bar{a}$, are averages where the bar denotes an exponential average over time. The relationship relates the next activation estimation to the present average, $\bar{a}_{(t)}$ and the present instantaneous activation, $a_{(t)}$, in an exponential manner, which is a recurrence relationship. The activation value is then passed through an Activation-output Function (which may be linear or sigmoidal) in order to produce the output. The Analogue-Model (A-Model) RAV-TIN's instantaneous real-valued activation is:

$$a_{(t)} = \frac{1}{S_m 2^n} \sum_{\mu} S_\mu \prod_{i=1}^{n} (1 + \bar{\mu}_j z_j) \tag{1}$$

hence it is term a sigma-pi model where the input probability distribution, defines the probability of the input. The S-Model's instantaneous activation is calculated on every forward pass as

$$a_{(t)} = (S_\eta / S_m) \tag{2}$$

and the average real-valued activation is defined as a function $a(z_i, S_\mu)$ by:

$$\bar{a}_{(t+1)} = \lambda \bar{a}_{(t)} + (1 - \lambda) a_{(t)} \tag{3}$$

where λ determines the activation's rate of increase and decay rate, the output is a function of, $y = f(\bar{a})$ hence

$$y = \sigma(\bar{a}_{(t+1)}) \tag{4}$$

The average activation is then used to obtain an estimate $y^* = \sigma(\bar{a}_{(t+1)})$ of the output y in the case of a sigmoidal Activation-output Function or $y^* = (\bar{a}_{(t+1)})$ in the case of a linear Activation-output Function. This in turn defines a distribution on a Boolean random variable, y, $P_1(\underline{y}) = y^*$, which is used to communicate the output to the next layer. Under stationary conditions, the output bits generated in this manner then estimate the value of for this node, if they are stored in a bit-stream. Then y^* may be estimated by

$$\underline{y}^* = (N_1/L) \text{ or } \bar{y}^* = 2(N_1/L) - 1 \tag{5}$$

where is the number of "1's" in a bit-stream of length L, $\underline{y}^* \in [0, 1]$ is the unpolarised estimate of the output and $\bar{y}^* \in [-1, 1]$ is the polarized estimate of the output. The sigmoidal, squashing, or logistic function $\sigma(.)$ is defined by

$$\sigma(\bar{a}) = \frac{1}{1 + e^{-\frac{(\bar{a})}{\rho}}} \tag{6}$$

The RVA-TIN may be utilised for continuous valued inputs where the real valued input defines the probability (z_i) of entering a "1" onto an input x_i of a node. One should note in the depiction of the RVA-TIN the site-values are quantised and the activations are quantised, hence the sigmoid is quantised into multi-levels, but this is due to quantisation of $\bar{a}_{(t+1)}$ to allow these models to be hardware realised. When the RVA-TIN sigma-pi unit is configured into a multi-layered net topology a change in the input probabilities means that the new outputs at the final layer will be estimated after L time steps, where L is the length of the output bit-stream. The sigmoidal function may not be required if, for example, one were using a RVA-TIN to perform an approximation to a function, e.g. a polynomial function, and if one required a linear unit one would utilise a linear Output Function.

4 Backpropagation of Error with instantaneous dynamic weights

In this section I define an implementation of Backpropagation that has its root in the original research carried out by Gurney[10], however I have developed the algorithm so it is faster and simpler. One should note that this work was not possible until Milligan[19] introduced a weight model for these units. The research into Back-Propa-

gation of errors for sigma-pi units of Gurney[10] was derived from the standard semi-linear Back-Propagation of Werbos[20] and then Rumelhart, Hinton and Williams[1]. This is based on the back propagation of a delta term which is derived from the previous node's output error times the weight of the connection associated with it. Gurney's breakthrough was the interpretation of these connection weights as probabilities which he derived from bit-streams, to provide what he terms 'dynamic weights'. I however do not use Gurney's bit-stream methodology to derive the weights on the connections of the nodes as bit-streams may have associated problems. Instead I define an instantaneous dynamic weight that may be derived for the present forward pass and does not have to be assimilated over multiple Feedforward operations as was previously the case with Gurney's dynamic weights derived from bit-streams. By utilising these dynamic weights the Back-Propagation paradigm cf the semi-linear units can then be directly interpreted into a sigma-pi notation of digital sigma-pi (probabilistic) logic nodes. The benefits of utilising the sigma-pi nodal model with this regime is that it uses stochastic state space search methods and the nodal model itself has an inherent ability to introduce noise while it is being trained. The Backpropagation training regime is dependent on the output error term which is defined as

$$e = \sum_{t=1}^{T} e_{(t)} = \sum_{t=1}^{T} \sum_{i=1}^{n} \frac{1}{2}(y_t^i - y^i)^2 \tag{7}$$

where y_t^i was the target output value. The actual output, y^i is defined as an expectation that is assimilated in an output bit stream of length L. The training algorithm presented first carries out L Feedforward passes to fill the bit stream. On the L-1 Feedforward pass the instantaneous dynamic weights are calculated, the output error or delta term. The weight update made at the penultimate Feedforward pass, utilising the instantaneous dynamic weight as

$$\Delta S_\mu^j = \alpha\delta^j \tag{8}$$

where the delta change is made to the jth unit in the net. For visible or output units the delta change to the jth unit in the output given input address μ is

$$\Delta S_\mu^j = \alpha(y_t^j - y^j)\sigma'(\overline{a^j}) \tag{9}$$

For hidden units the delta change to the jth unit in the hidden layer, given input address μ is

$$\Delta S_{\mu}^{j} = \alpha \frac{2}{S_m} \sigma'(a^j) \sum_{k \in I_j} \delta^k \bar{w}_{i(j)}^{k} \tag{10}$$

where node k is the fan-out unit of node.The instantaneous dynamic weights w_i as

$$\underline{w}_i = \frac{S^k_{\mu[1_i]} - S^k_{\mu[0_i]} + 2S_m}{4S_m} \tag{11}$$

where $\mu[/_i]$ be the n-tuple partial *address* formed by placing a "don't care" or wild card in the ith bit position in $\mu \in \{\mu_1, \mu_2, \mu_3, \ldots \mu_i\}$. This in fact means that each input line to the cube or n-tuple is endowed with the ability of toggling its binary value and hence toggling the address this defines e.g. if for example the second bit is toggled in a 3-tuple unit one obtains two partial addresses ($\mu[/_2]$) i.e. $\mu_1,\mu_2,\mu_3 \rightarrow \mu_1, \neg\mu_2,\mu_3$, where $\neg\mu_i$ implies an inverse or complement of the bit. Hence $\mu[1_i]$ set the ith bit position in address μ to 1 and $\mu[0_i]$ set the ith bit position in address μ to 0 in order to calculate the ith instantaneous dynamic weight. To train networks of cubic nodes one requires polarized weights, because if the connection's weight has an inhibiting effect (i.e. it is a negative quantity.) then the Backpropagated delta term must reflect this. Hence

$$\bar{w}_i = S_m(2\underline{w}_i - 1) \tag{12}$$

This is the polarised instantaneous dynamic weight $\bar{w}_i \in \{-S_m, S_m\}$.

5 Pre-calculated look-up tables for Backpropagation of Error training of the Quantised Real-Valued Activation S-Model

The method usually used to implement a Neural Net on a sequential computer is to first set up an array for the weights, then randomly initialise the weights sequentially and finally to train the net by adapting each of the weights in the net sequentially using a regime such as Back-propagation of Error. The method used to train the neural network utilises constrained look-up tables, Neville et al.[13], this enables one to pre-calculate the variables for the Feedforward (FF) phase and while carrying out the last (L-1) FF operation one can download the pre-calculated partial delta changes, $\alpha\sigma'(\bar{a})$ for the output units and $\alpha(2/S_m)\sigma'(\bar{a})$ for the hidden units, at the same time. This may

be done because the sigma-pi neuron model may view its activation in the polarized notation as $\bar{a} \in \{-1, \ldots, 1\}$ and these may be interpreted in turn as a set of integer elements $\bar{a}_q \in \{-\bar{a}_m, \ldots, \bar{a}_m\}$ which are represented in the machine as a set of integer elements denoted by '*m-q*' the machine-quantised representation, $\bar{a}_{m-q} \in \{0, \ldots, \bar{a}_m, \ldots, 2\bar{a}_m\}$ e.g. for $\bar{a}_m = 5$ then $\bar{a}_{m-q} \in \{0, \ldots, 5, \ldots, 10\}$. This enables one to pre-calculate constrained look-up tables which the net requires for the FF phase and the training phase. This may be best described in the following manner. Given the pre-calculated partial delta changes, $\alpha\sigma'(\bar{a})$ for the output units and $\alpha(2/S_m)\sigma'(\bar{a})$ for the hidden units, where the constant $0 < \alpha$ is the learning rate, which is initialised prior to the training phase. The output delta term $\delta^j = (y_t^j - y^j)$ is the real-valued output error of the *j*th sigma-pi unit. The variables $\sigma(a^j)$ and $\sigma'(a^j)$ take on one of $D = 2\overline{a_m} + 1$ discrete values. An example pre-calculated FF and training constrained look-up table is depicted in Table below:

Binary $\bar{a}_{m-q}$	$\bar{a}_{m-q}$	$\sigma(\bar{a}_{m-q})$	$\sigma'(\bar{a}_{m-q})$	$\alpha\sigma'(\bar{a}_{m-q})$	$\alpha\frac{2}{S_m}\sigma'(\bar{a}_{m-q})$
000_2	-2	0.09	0.08	0.17	0.35
001_2	-1	0.25	0.18	0.37	0.74
010_2	0	0.50	0.25	0.5	1.0
011_2	1	0.75	0.18	0.37	0.74
100_2	2	0.9	0.08	0.17	0.35

Table 1: pre calculated and training constrained look-up table

In the Table, $\overline{a_m} \in \{-2, \ldots, 2\}$ and ρ=0.45, $S_m = 1$ and $\alpha = 2.0$ for the sigmoidal output case. Note that one may download all the partial $\Delta S_{\mu_{m-q}}$'s in the FF phase. At the end of the FF phase the partial $\Delta S_{\mu_{m-q}}$ is used to update $S_{\mu_{m-q}}$, as the output error is calculated after the FF phase and each node's output Y^j is known after the FF phase. This is carried out by multiplying the partial $\Delta S_{\mu_{m-q}}$ by the real-valued delta term δ to obtain the actual $\Delta S_{\mu_{m-q}}$. The pre-calculated look-table approach is dependent on the number of discrete activation states, $D = 2\overline{a_m} + 1$.

6 Experimental Work

In order to illustrate the learning accuracy of the RVA-TIN, simulation results are presented for units/nets trained on three different experiments. The tasks are either multiple input or single input and single output. Task 1 maps the XOR function, $\{x_1, x_2\} \rightarrow \{y\}$, $\{0.1,0.1; 0.1,0.9; 0.9,0.1; 0.9,0.9\} \rightarrow \{0.1,0.9,0.9,0.1\}$ using a two-input hidden node and a three-input visible node. Two inputs to the output node are

connected to the two hidden node inputs and the third is connected to the output of the hidden unit. Task 2 maps the function $x \rightarrow x^2$, using a single layer eight input unit. Task 3 maps the function $x \rightarrow x^2$, using a network made up of eight three-input hidden units and an eight-input visible unit. The function is mapped using twelve input output stimuli-pairs spaced equidistant across the function $x \in [0, 1]$. One should note in task 2 each of the eight-inputs and task 3 each of the three-inputs of the hidden units were allocated the same analogue input value. Each has its own random number in order to obtain independent instantaneous binary inputs. This is necessary in order to obtain multiple binary inputs, if this is not done only addresses zero and the maximum input address are selected and the cube effectively becomes a 1-cube. Hence to obtain input addresses that utilise the full range of cube locations each input line's input bit is generated independently of the others For all the experiments the constants utilised were $\alpha = 32.0$, $\rho = 24.0$, $\lambda = 0.75$, $L = 1024$, $S_m = 64$ and $\overline{a_m} = 128$.

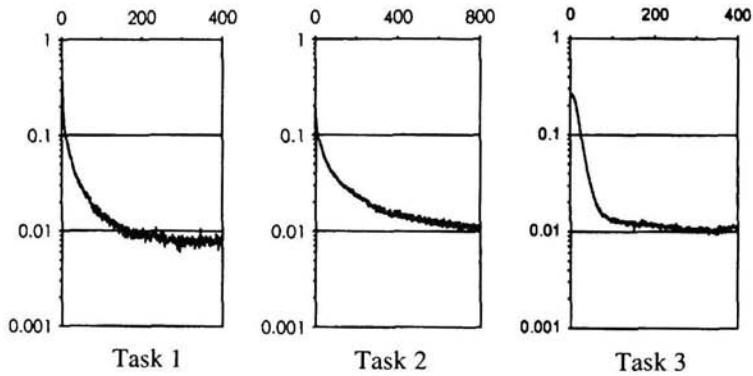


Figure 2: Error modulus over 20 nets for various training tasks.

The graphs above show the average error modulus over 20 nets for a given training epoch versus the number of epochs. Each epoch is defined as the presentation of all four/twelve input-output stimuli pairs. The graph of Task 1(XOR) shows that the majority of the nets converged to an error modulus of 0.01 in less than 200 epochs. Task 2($x \rightarrow x^2$) mapping the squared function is shown to be a harder task and takes at least 800 epochs for the single layer nets to converge to an error modulus of 0.01. However by introducing a hidden layer in Task 3($x \rightarrow x^2$) the nets converge to an error modulus of 0.01 in 300 epochs, which shows the significance of a hidden layer with regards to expanding the internal state space of the system.

6.1 *Comparison with other methods*

For comparison purposes only the following paragraph discuss the speed of convergence of our RVA-TIN units trained with Back-propagation and the SRV (Stochastic Real Valued) units of Gullapalli[21,22] trained with deterministic Reinforcement. Of course, one should note that if Gullapalli utilised a supervised regime he would have faster learning, but by utilise the same task as he does in his simulations we can compare the convergence times to show how a supervised methodology is more efficient. I also compare our RVA-TIN units trained with Back-propagation and the semi-linear Threshold Logic Units (TLU) of Gurney[10] trained with System Identification. The results of the three tasks are tabulated below to show the learning efficiency of the RVA-TIN units when compared to SVR units and TLU units.

Table 2 shows a comparison of learning efficiency of the RVA-TIN units when compared to SVR units and TLU units when tested on the three tasks.

Task #	RVA-TIN Epochs (*lel*)	SVR Epochs (*lel*)	TLU Epochs (*lel*)	RVA-TIN Min. *lel*
1	75 (0.01)	2400 (0.01)	-	0.00122
2	418 (0.01)	-	1500 (0.063)	0.00487
3	77 (0.01)	-	1500 (0.063)	0.006

Table 2: Comparison of RVA-TIN, SVR and TLU units

The above table shows that Backpropagation training of RVA-TINs is more efficient that either of the others regimes. However we are not presenting the results in light of their efficiency, but just as comparisons. The more important point to note is that RAM- based sigma-pi nets allow one to Partially Pre-calculated weights for Back-propagation training.

7 Concluding Remarks

The chapter presents new techniques which enable real-valued functions to be mapped onto RAM-based RVA-TIN sigma-pi nets utilising Backpropagation training. The study shows that one may utilise quantised weights (site-values, $S_\mu \in \{-64, \ldots, 64\}$) which are stored in 8-bit binary code and quantised activations ($\tilde{a} \in \{-128, \ldots, 128\}$) which are stored in 9-bit binary code and map real-valued function to an accuracy of 1%. The research work makes use of the quantised nature of the RVA-TIN nodes activation, to calculate the sigmoidal activation-output func-

tion and the sigma primed terms, by utilising look-up table methodologies, one of the problems inherent in most neural networks which have very serious overheads relating to complex and time consuming mathematical function calculations which are normally calculated during the feed-forward and training phases of these neural systems. This research enables sigma-pi structures to be efficiently trained and mapped on to RAM-based technologies. By using a methodology of constrained look-up tables one can pre-calculate the sigma-pi's activation-output function, $\sigma(\bar{a})$ and the partial weight (site-value) updates. This means that computationally intensive mathematical functions, such as the sigmoidal function (6), are pre-calculated. Previous work using semi-linear units utilising Backpropagation of Error training (Grozinger, [10]) utilise an approximated sigmoid, using piece-wise linear functions. The RVA-TIN sigma-pi has bounded activations and site-values which aid the high speed training of a neural system as one implements look-up tables for the majority of the mathematical computations. Due to the functionality of sigma-pi units one does not carrying out the more normal $\sum x_i w_{ij}$ calculations that are used by semi-linear units, which require one to perform n multiply operations (where n is the number of inputs to node j.) and then sum all the products, these are time consuming tasks, hence the computational requirements of sigma-pi units are less than semi-linear.

To put this research in perspective and to counter balance the above comments one does have to provide storage space for the pre-calculated look-up tables to store $\sigma(\bar{a})$, $\sigma'(\bar{a})$, $\alpha\sigma'(\bar{a})$ and $\alpha(2/S_m)\sigma'(\bar{a})$. In the experiments $\overline{a_m} = 128$ this means that the system requires $4 \times (2\overline{a_m} + 1)$ memory locations which is approximately $1k$ of memory to implement the pre-calculated look-up tables. But when one utilises large artificial neural networks this becomes an insignificant requirement when one is now able to pre-calculate complex functions which do not have to be calculated in real time while the system is training.

The use of RAM-based Sigma-pi nets is not wide spread, however in this chapter I have presented research that shows they have some advantages over the more commonly utilised semi-linear units. The major advantage they have is that they are implementable in cheap RAM-based hardware. It is interesting to note that one can now obtain 4Gb dynamic RAM chips. However they may also be implemented on Massively Parallel Processors (Neville et al.[13]).

References

1. D.E. Rumelhart, G.E. Hinton, G.E. Williams, Learning Internal Representations by Error Propagation, *Parallel Distributed Processing*, Volume 1, The MIT Press, USA, 1986.
2. B. Lenze, How to Make Sigma-pi Neural Networks Perform Perfectly on Regular Training Sets, *Neural Networks*, Volume 7, Number 8, pp. 1285-1294, 1994.
3. I. Aleksander, T.J. Stonham, Guide to pattern recognition using random-access

memories, *Computers and Digital Techniques*, 2, pp 29-40, 1978.

4. B.A. Wilkie, A stand-alone, high resolution, adaptive pattern recognition system, *Ph.D Thesis* W499, Department of Electrical Engineering, Brunel University, Middlesex, UK., 1983.
5. W.-K. Kan, A probabilistic logic neuron for associative learning. In *Proceedings of the International Conference on Neural Networks*, IEEE, ICCN'87, San Diego, CA, USA, pp. 541-548, 1987.
6. B. Zhang, L. Zhang, and H. Zhang, A Quantitative Analysis of the Behaviours of the PLN Network. *Neural Networks*, Volume 5, Number 4, ISSN 0893-6080, pp 639-644, 1992.
7. J. Austin, (1994). A review of RAM based neural networks, *Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems*, September 26-28, Turin, Italy, ISBN 0-8186-6710-9, pp 58-66. See Also Chapter 1 of this book.
8. I. Aleksander, Neural Computing Architectures: The design of brain-like machines. Ed. by Aleksander, I., *North Oxford Academic Publishers Ltd*, ISBN 0-946536-47-3, re. The logic of connectionist systems, Chapter 8, pp. 133-155. and A probabilistic logic neuron network for associative learning, Chapter 9, pp 156-171 (see also Chapters 1, 10 & 11.), 1989.
9. D. Gorse, J.G. Taylor, Reinforcement Training Strategies for Probabilistic RAMs, Theoretical Aspects of Neurocomputing: Selected papers from the *Symposium on Neural Networks and Neurocomputing*, pp 180-184, 1990.
10. K. Gurney, Learning in nets of structured hypercubes. *PhD thesis*, Department of Electrical Engineering, Brunel University, Middlesex, U.K. available as: Technical Memo CN/R/144, 1989.
11. R.S. Neville, Investigate and Evaluate the Design of Probabilistic Nodes for Boolean n-cube Networks, *MSc Thesis*, Department of Electrical Engineering, Brunel University, UK, 1990.
12. P.J. Werbos, Beyond regression: new tools for predictions and analysis in the behavioural sciences, *Ph.D. dissertation*, Committee on Applied Mathematics, Harvard University, Cambridge, USA, 1974.
13. R. Neville, R.J. Glove, J. Stonham, Evaluation of Training & Mapping sigma-pi Networks to a Massively Parallel Processor, IEEE ICNN'95, *International Conference on Neural Networks*, Perth, Western Australia, 27 November - 1 December, pp 1042-1047, 1995.
14. M. Grozinger, Feasibility of Neurocomputing on the Associative String Processor. *Master's thesis*, Department of Electrical Engineering, Brunel University, Uxbridge, U.K., 1990.
15. R.S. Neville, T.J. Stonham, Adaptive Associative Reward-Penalty Algorithms for sigma-pi Networks, in: *Neural, Parallel & Scientific Computations*, Vol. 2,

No 2, June, ISSN 1061 5369, pp. 141-164, 1994.

16. R.S. Neville, T.J. Stonham, Generalisation in sigma-pi Networks, in: *Connection Science*, Volume 7, Number 1, ISSN 0954 0091, pp. 141-164, 1995.
17. R.S. Neville, T.J. Stonham, Adaptive Critic for sigma-pi Networks, *Neural Networks*, Volume 9, Number 4, June, ISSN 0893-6080, pp. 603-625, 1996.
18. A. Barto, R. Sutton, C. Anderson, Neuron like adaptive elements that can solve difficult learning problems. In *IEEE Transactions on systems, man, and cybernetics*, SMC-13(5):834-846, 1983.
19. D.K. Milligan, Annealing in RAM-based learning networks. *Brunel University Technical Memorandum CN/R/142*. Middlesex, U.K, 1988.
20. P.J. Werbos, Beyond regression: new tools for predictions and analysis in the behavioural sciences, *Ph.D. dissertation*, Committee on Applied Mathematics, Harvard University, Cambridge, USA, 1974.
21. V. Gullapalli, (1988). A stochastic algorithm for learning real-valued functions via reinforcement feedback. *COINS* Technical Report Number: 88-91, September 29th, The Department of Computer and Information Sciences (COINS), The University of Massachusetts, USA.
22. V. Gullapalli, (1990). A Stochastic Reinforcement Learning Algorithm for Learning Real-Valued Functions. *Neural Networks*, Volume 3, Number 6, pp. 671 - 692.

OPTIMISATION OF RAM NETS USING INHIBITION BETWEEN CLASSES

T.M. JØRGENSEN
Risø National Laboratory, P.O. Box 49
DK-4000 Roskilde, Denmark

A strategy for adding inhibitory weights to RAM based net has been developed. As a result a more robust net with lower error rates can be obtained. In the chapter we describe how the inhibition factors can be learned with a one shot learning scheme. The main strategy is to obtain inhibition values that minimise the error-rate obtained in a cross-validating test performed on the training set. The inhibition technique has been tested on the task of recognising handwritten digits. The results obtained matches the best error rates reported in the literature

1 Introduction

With respect to the learning of classification tasks the conventional N-tuple nets or RAM-nets such as the WISARD system[1] have many nice properties such as fast training and recall rates. Furthermore, the architecture is very easy to interpret. With P classes to distinguish the set of N-tuples can be considered as a number of Look Up Tables (LUT). This is illustrated in figure 1. Each LUT probes a subset of the binary input data. The sampled bit sequence is used to construct an address. This address corresponds to a specific entry in the LUT. The number of output values (rows) from a given column entry is equal to the number of possible classes. For each class the output can take on the values 0 or 1. A value of 1 corresponds to a vote on that specific class. The output vectors from all LUTs are added, and subsequently a winner takes all decision is made to perform the classification.

The number of LUT-addresses shared by any two examples belonging to different classes should be as small as possible. However, in many situations two different classes might only differ in a few of their features. In such a case an example (that has not been used for training) has a high risk of sharing most of its features with an incorrect class. In this situation the RAM will have an unacceptable high error rate. In order to circumvent this situation it is desirable to weight different features differently for a given class.

The CMAC[2] is an example of an architecture that allows real weights to be stored in the RAM-net. These weights can then be trained with a perceptron like learning rule. We present an alternative solution introducing negative weights into the LUT cells. With this approach the modified architecture bears a close resemblance to the

conventional system. Accordingly, it also preserves some essential advantages of the simple RAM-net.

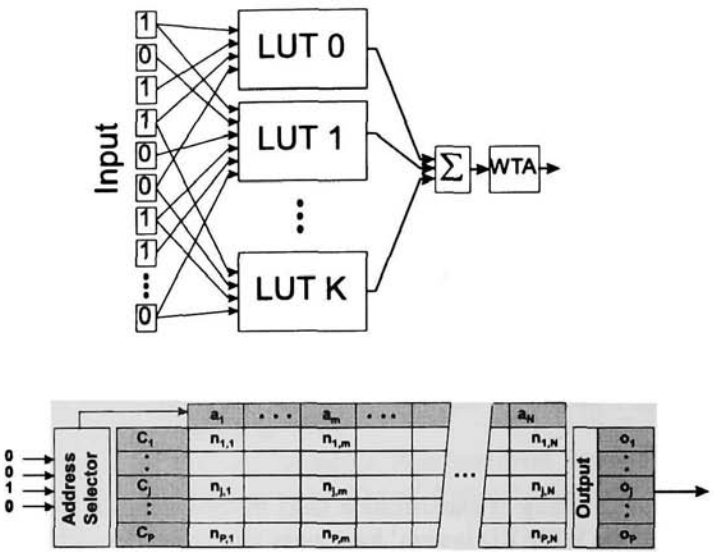


Figure 1: Architecture of the RAM net. The number of training sets from class c_j that generate address a_m is denoted by $n_{j,m}$. If $n_{j,m} \geq 1$, the output value o_j is set to 1, else 0.

2 The need for inhibition

In order to illustrate the need for inhibition we consider the case of distinguishing between the digits 4 and 9. The main difference between 4's and 9's is the appearance or non-appearance of a upper horizontal bar as illustrated in figure 2. If such a bar is met in a test example it is desirable that the network inhibits class 4. Otherwise there is a risk that too much emphasis is put on the other parts of the test pattern. In general there can be several LUT columns in the network that corresponds to variations of such a bar. These columns will be characterised by not "voting" on class 4 but possible on class 9. Due to the limited size of the training set it is, however, not necessarily so that columns voting on class 9 but not on class 4 (and vice versa) represent real distinguishing features. Accordingly, a strategy is needed for selecting those columns that are the most likely candidates for inhibition.

Figure 2: An example of two digit classes that are very close with exception of the bar (or non-bar structure) at the top.

3 Obtaining the inhibition values

The first step in locating column candidates for inhibition is to locate the training examples having low confidence as well as those being misclassified in a cross-validating test (see the accompanying chapter[3] on using cross-validation tests in connection with RAM nets). For each of these examples all LUT columns voting on the true class but not on the competing class are found. Afterwards a small inhibition term, $-\alpha_{inhib}$, is added to the weight value of the competing class for each detected column. A typical value of α_{inhib} is $10/N_{LUTs}$, where N_{LUTs} denotes the total number of LUTs in the architecture.

The inhibition factor is calculated so that the confidence after inhibition corresponds to a desired level. Inevitably this technique will add inhibition to some LUT columns that do not represent relevant distinguishing features. However, the LUT columns being sought are likely to be visited by many of the low confidence training examples and accordingly their corresponding cells will obtain larger inhibition factors than the rest. From the argumentation it is also evident that α_{inhib} should be sufficiently small, in the sense that the effect of inhibiting "wrong" cells will become negligible.

An advantage of using the above described inhibition technique is that essential features of the basic RAM net model are preserved:

- It is guaranteed that each training pattern always obtains the maximum number of votes. As a result the network makes no misclassification on the training set (ambiguous decisions might occur).

- The inhibition scheme is a one-shot learning scheme.

All output values that would be set to 1 in the simple system are also set to 1 in the modified version. However, some of the cells containing 0's in the simple system will have their contents changed to negative output values in the modified net. In other words, the conventional net is extended so that inhibition from one class to another is allowed.

The traditional RAM net architecture can also be viewed as a constrained feed-forward architecture. This is illustrated in figure 3. Every LUT column with non-zero weights corresponds to a neuron in the hidden layer. As only one column pr. LUT can be addressed at a given time the number of active neurons in the hidden layer is limited to the number of LUTs. As shown in figure 3 the traditional architecture does not allow any inhibition from the hidden units to the output units, but such inhibitory weights are introduced by the above described technique.

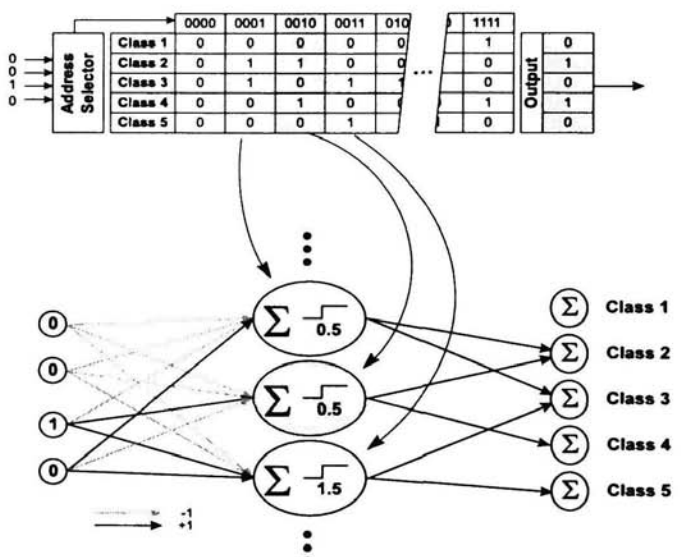


Figure 3: The figure illustrates how the RAM net architecture can be viewed as a constrained feed-forward architecture. The three neurons shown in the hidden layer correspond to three columns of the depicted LUT. Introducing inhibition between classes corresponds to allowing negative weights on the connections between hidden neurons and the output layer.

4 Results

We have tested the inhibition scheme on the problem of recognising handwritten digits. This specific task is often used to evaluate and compare different classifiers (neural networks, statistical based classifiers and hybrid configurations)[4,5,6]. We trained and tested our neural networks on digits from the NIST database[7], which has become a standard database to use for this task. The digits were scaled to a 16×16 format and centred. No other pre-processing was performed.

For selecting input connections the most simple strategy is to pick the input connections at random. However to increase the probability of detecting local features we have chosen a scheme where a large part of the LUTs picks their connections within local receptive fields of size 4×4. Within these receptive field areas the input connections are chosen at random. The appropriate numbers of input connections per LUT are found using the technique described in the accompanying chapter[3]. The obtained results are listed in Table 1. The cross-validation and cogentropy methods for selecting LUT connections (as opposed to a random selection scheme) are also described in the accompanying chapter.

From the results it can be seen that the added inhibtory weights improve the performance of the RAM network. Actually the error rate has been reduced by almost 60% for the large network with 935 LUTs. The obtained error rate of 1.2% matches the best results reported in the literature[8].

Number of examples		Number of LUTs	Number of inputs pr. LUT	Training method	Error rate	
Training	Test				without inhibition	with inhibition
14095	13521	200	13	cogentropy	3.1%	2.3%
14095	13521	200	13	CV	3.4%	2.6%
14095	13521	200	13	random	6.0%	5.1%
14095	13521	200	13	cogentropy	3.1%	2.3%
14095	13521	935	10-14	random	2.8%	1.2%

Table 1: Test results obtained for the task of recognising handwritten digits.

In order to get an idea of the impact of the inhibition techniques on the architecture we evaluated the number of negative weights that were added for the architecture with 935 LUTs. In total around 36000 negative weights have been added to the cells. This corresponds to an average around 40 negative weights pr. LUT. For comparison the number of positive weights (all having value 1) is 1.8 million in total (around 2000 pr. LUT). The number of inhibitory weights is seen to be small compared to the number of positive weights; nevertheless they have a large impact on the performance.

As described in Jørgensen[3] it is possible to calculate what we denote a critical example number, n_{crit}, which estimates the amount of examples from the training set that

support a given classification. The n_{crit} value is suited as a confidence measure, and accordingly it can be used as a part of or as *the* rejection criterion. This is illustrated in figure 4. (the curves correspond to the RAM net with 935 LUTs listed in table 1). The upper curve shows the amount of errors that are accepted for different acceptance levels of n_{crit}, whereas the lower curve shows the amount of accepted classifications among the examples being correctly classified. As an example it can be seen that using an n_{crit} value of 2 as acceptance level implies an error rate of 0.1%.

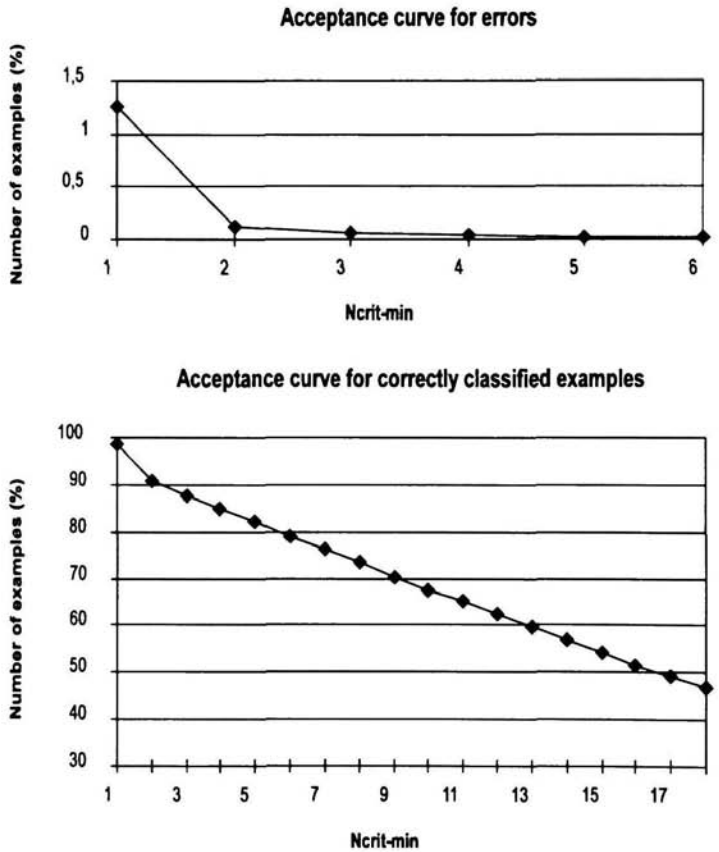


Figure 4: The two curves illustrate the amount of errors and correctly classified test examples that are accepted for different acceptance levels of n_{crit}.

5 Conclusion

A one shot learning technique that introduces inhibition into a RAM net architecture has been described. This new technique was tested on the task of recognising handwritten digits, where it leads to a significant performance improvement. The obtained error rate matches the best results reported in the literature. The obtained error rate (without rejection) on a test set of 13521 digits was 1.2%.

Acknowledgments

I thank Steen Sloth Christensen for stimulating discussions and for providing me with his basic software implementation of a RAM neural net.

References

1. I. Alexander and T.J. Stonham, *Computers and Digital Techniques*, **2**, 29-40 (1979).
2. J. S. Albus, Journal of Dynamics System, Measurement, and Control, Transaction of the ASME, **97**, 220, (1975).
3. T. M. Jørgensen, S. S.Christensen, and C. Liisberg, Crossvalidation and information measures for RAM based neural networks, (accompanying chapter).
4. Z. Chi and H. Yan, Neural Networks, **8**, 821, (1995).
5. C.L. Wilson in *SPIE Proceedings* Vol. 1906, ed. Donald P. D'Amato (1993).
6. S. Lee, Neural Networks, **8**, 783, (1995).
7. National Institute of Standards and Technology: NIST Special Data Base 3, Handwritten Segmented Characters of Binary Images, HWSC Rel. 4-1.1, (1992).
8. J. Geist et al., Technical Report NISTIR 5452, (1994).

A NEW PARADIGM FOR RAM-BASED NEURAL NETWORKS

G. HOWELLS, D.L. BISSET, M.C. FAIRHURST
Electronic Engineering Laboratories,
University of Kent,
Canterbury, Kent, CT2 7NT, UK

This chapter introduces a novel networking strategy for RAM-based Neurons which significantly improves the training and recognition performance of such networks whilst maintaining the generalisation capabilities achieved in previous network configurations. A number of different architectures are introduced each using the same underlying principles.

1 Common Architecture Features

Initially, features which are common to all architectures are described illustrating the basis of the underlying paradigm. Three architectures are then introduced illustrating different techniques of employing the paradigm to meet differing performance specifications.

2 Overview of Network Structures

This chapter will introduce a family of neural architectures which collectively encompass a basic set of common attributes. The network architectures introduced are all RAM-based networks employing layers of neurons each of which is independently attached to a given sample pattern whose distinct outputs are merged (by means of a further layer) to produce an output matrix which is equal in dimension to the original sample pattern. Each architecture may employ a varying number of groups of such layers arranged sequentially, and a general structural pattern for a group is shown in Figure 1. Each of the architectures introduced in this chapter share the following common features:-

- The neurons comprising the network are arranged in $x \times y$ matrices or layers where x and y are the dimensions of the patterns under consideration.

- Each element within the pattern is therefore associated with a corresponding neuron within each layer.

- The layers comprising the network are arranged in a fixed number of groups, the exact number of such groups being architecture dependent.

- A **Merge** layer exists after each group whose function is to combine the corresponding outputs of the constituent layers of the group. The connectivity of the neurons comprising a **Merge** layer is equal to the number of layers within the group to which it pertains.

- The number of layers within each group may be varied depending on the recognition performance required from the network.

- The constituent layers of a group differ in the selection of elements attached to the inputs of their constituent neurons (termed the *connectivity pattern*).

- Neurons within a given layer possess the same connectivity pattern relative to their position within the matrix.

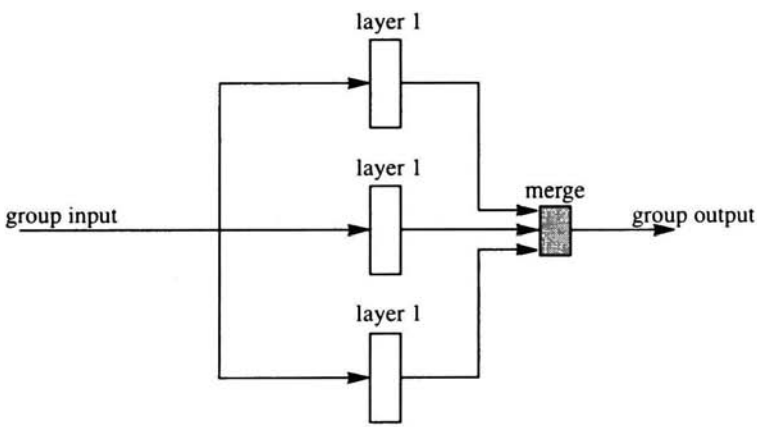


Figure 1: A Group of layers

The connectivity pattern for neurons of differing layers can take various forms. Since, for each neuron, the input set is calculated modulo the dimensions of the pattern, then neurons which, for example, are situated at the right edge of a pattern and are virtually clamped to the 'x' bits on their right will actually be clamped to 'x' bits

at the left side of the pattern.

The layers within the network are all modified independently during the training phase and the precise training algorithm is dependent on the architecture under consideration.

The architectures presented below all conform to the basic principles outlined above with modifications to the number of groups and layers within a group as required. They differ however in the structure of the neurons employed within the constituent layers of the architecture. Although all are based on RAM-based neurons, the number of symbols employed by the neurons differs between the architectures and the following sections will describe and discuss the differences occurring in different architecture.

3 The Boolean Convergent Network (BCN)

The Boolean Convergent Network (BCN)[1] is the most basic network architecture in the series and the structure from which other architectures were developed. It is a RAM-based neural network where the inputs and output of all the component neurons are taken from the symbols '0','1' and the undefined value 'u'. The inputs to a neuron form an addressable set incorporating all memory locations which may be formed by treating any undefined value within the input as either a '0' or a '1'. The output of a neuron will be any defined value which occurs exclusively within the memory locations included within the addressable set. If the addressable set contains either no defined value or examples of both defined values, then the undefined value 'u' is output.

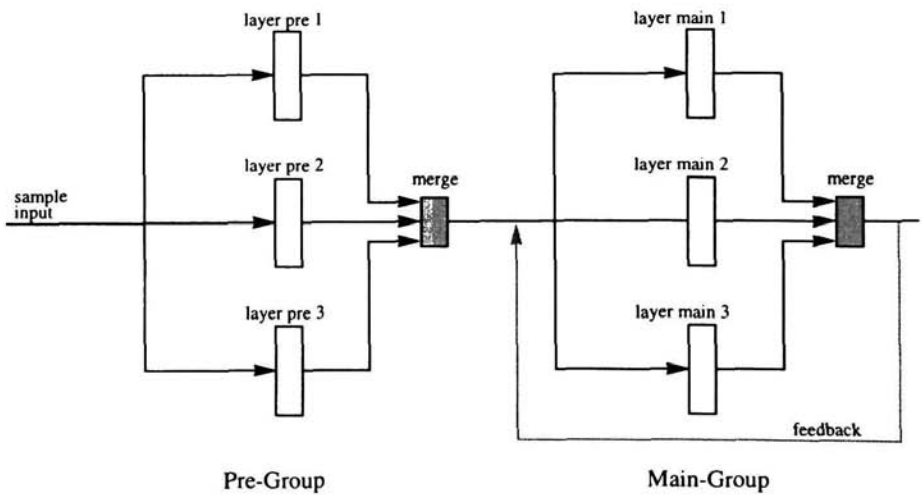


Fig 2: Example of BCN Network Architecture

An example of the general structure of the basic BCN architecture is illustrated in figure 2. and it can be seen that it employs three groups of layers, each of which possesses the properties already described above.

Although the training and recognition algorithms relevant to BCN are simple and easy to implement, it should be noted that any neuron can produce only one of the three symbolic outputs '0', '1' or 'u'. Therefore, for any dataset containing more than three pattern classes, for each given neuron, there must exist examples from different pattern classes which map to the same output symbol. The problems surrounding this overloading will be examined below.

3.1 Performance of BCN

The performance of the BCN architecture in a typical pattern classification task, where the sample patterns were taken from digitised postcode data corresponding to the numerals 0 through 9 with ten example patterns of each numeral presented to the network for training, is shown in Table 2. Three hundred sample patterns of each numeral were presented for processing. The same training and sample patterns were used for all the experiments represented.

The various columns indicate the performance achieved by varying the number of layers within the groups. Two differing connectivity patterns for the 13 layer option are shown (labelled 13A and 13B). It is clear that there is scope for optimisation of the network in terms of the number of layers and set of connectivity patterns for a given dataset.

These results demonstrate the potential classification performance of such a network, even though the experiments were not exhaustive.

Class	Layer Variations				
	6	9	13A	13B	17
0	92	95	99	99	98
1	78	78	80	82	83
2	92	97	91	91	90
3	93	97	96	95	94
4	95	96	95	96	94

Table 1: Performance of BCN with variations shown (% correct)

Class	Layer Variations				
	6	9	13A	13B	17
5	75	74	75	73	70
6	77	86	84	83	83
7	95	96	96	96	95
8	93	91	95	95	96
9	85	91	91	89	94
average	87.5	90.1	90.2	89.9	89.7

Table 1: Performance of BCN with variations shown (% correct)

4 The Generalised Convergent Network (GCN)

Although the BCN architecture is simple to implement and train, it is, in practice, suitable only for processing of datasets in which the constituent classes are relatively tightly clustered as in the case, for example, in the processing of machine printed character datasets. More complex datasets involving handwritten characters demand a higher performance specification and for this it is necessary to consider how the BCN architecture may be improved. In order to identify how to achieve this, it is first necessary to identify precisely where its deficiencies lie.

A major problem with the BCN architecture, and indeed a common problem with Boolean RAM-based networks, is that a particular neuron output represents a set of pattern classes when the input to a neuron specifically indicates a particular class. For example, a neuron may output the value '0' when an exact match is found with either of two separate pattern classes. The '0' output does not indicate which of the two pattern classes has been identified and, more worryingly, is indicating that the input is consistent with a second pattern class when in fact it is not. This may lead to increased difficulty in recognition tasks and increase the probability of incorrect recognition.

The problem arises because neuron outputs are restricted to a small set of symbols, often just '0' and '1'. These symbols must then be overloaded in order to represent all the pattern classes under consideration. It is possible to remove the overloading by increasing the symbol set available to the neurons. This may be achieved in the following two ways.

Firstly, the symbol set may be extended to allow a symbol to represent each pattern class under consideration. For example, if ten pattern classes representing numerals were being considered, the symbols '0' through '9' could be employed

representing the numerals 0 through 9 respectively. These symbols may be referred to as the *base* symbols of a network.

This general arrangement will remove the problem of neuron outputs matching multiple arbitrary pattern classes, the symbol pertaining to the required class may be used, but may still allow neurons to lose information. For example, if a neuron is presented with identical inputs for examples of two differing classes, classes 1 and 2 say, it has no way of indicating that it has an input consistent with classes 1 and 2 but not with other classes. It must indicate either class 1 or class 2 or neither class. It other words, it is sometimes desirable to produce a value which is consistent with multiple pattern classes, specifically when the input is also consistent with these same pattern classes.

The problem may be alleviated by allowing yet further symbols to indicate where combinations of pattern classes are allowed. To use the example already cited, if a neuron address is consistent with examples of classes 1 and 2, it should store a single symbol representing both classes 1 and 2.

In general, extra symbols are required for each possible combination of pattern classes. The number of symbols required will thus be equal to the cardinality of the powerset of the set of pattern classes. These symbols, which include the base symbols, may be referred to as the *compound* symbols of the network.

An architecture employing neurons incorporating the above proposals has been developed referred to as the Generalised Convergent Network $(GCN)^2$ because of the generalised nature of the symbols employed. In abstract form, the GCN architecture is similar to that of BCN shown in figure 2 except that only two groups (termed the **Pre** and **Main** groups) are employed.

4.1 Performance of GCN

The Performance of the GCN architecture has been examined using a number of datasets. Some typical results are shown in Table 1. The results were achieved using an architecture employing 9 layers within both the **Pre** and **Main** groups with all neurons of connectivity 8.

The first results column shows the performance achieved using 300 sample machine printed characters per class with only 10 training patterns employed for each class. It thus provides a comparison with the BCN results shown in figure 2.

The second and third columns indicate results achieved with handwritten characters using two sample datasets respectively. The results were achieved using a varying number of training and sample patterns per class (the entire dataset was employed) and illustrate the potential of the network to perform handwritten character recognition.

The final two columns indicate the results achieved using a standard handwritten

dataset with firstly the entire training set (C1) and secondly the same training set with potentially confusing training patterns manually removed (C2). The results thus illustrate the potential improvement expected where it is possible to optimise a training set.

Class	Machine	Handwritten			
		A	B	C1	C2
0	98	94	89	93	97
1	98	88	72	61	61
2	99	74	94	98	100
3	98	87	80	87	90
4	99	90	95	92	95
5	99	80	64	83	86
6	94	75	73	90	94
7	99	65	79	77	81
8	99	96	66	97	98
9	99	76	72	88	91
average	98.0	82.5	78.4	86.6	89.3

Table 2: Performance of GCN (% correct)

5 The Probabilistic Convergent Network (PCN)

The GCN architecture produces a much improved performance over that of the simpler BCN architecture at the expense of an increased complexity. It does however produce a simple 'recognition' of a sample pattern. It provides no indication of how certain it is of its conclusion or of how closely the sample pattern matches other pattern classes. For some applications, this information may be required (humans after all can provide this information when considering patterns). To provide this capability, a further architecture enhancement is necessary. As was done when developing GCN, an examination of where the deficiencies lie within the GCN architecture is performed in order to devise where further modifications may be made.

A given storage location within a **Pre** group GCN neuron indicates which training sets (and therefore which pattern class) contain example patterns which generate

an address corresponding to the neuron within the training process. For example, if examples of classes 1 and 2 addressed the given location within the neuron, the location would contain the symbol '12' correctly indicating that if a sample pattern addressed the location it would be consistent with it being an example of pattern class 1 or 2. However, it does not indicate what proportion of the members of the respective training sets addressed each location. For example, almost all the members of the training set for class 1 may address the location while only one of the members of the training set for class 2. The symbol stored in the location would still be '12' even though a sample pattern addressing this location would be more likely to be an example of a member of class 1 than of class 2. The question is "Is there a method of allowing a neuron to indicate that a sample pattern is more likely to be a member of class 1 while still allowing the possibility that it is a member of class 2 if other neurons take this view?" The Probabilistic Convergent Network (PCN) has been designed to achieve this aim.

The Probabilistic Convergent Network (PCN)[3] extends from the architectures described above but still retains the same abstract structure as BCN and GCN as was described earlier. Two groups of layers are employed by the network with a feedback for the **Main** group. There are a number of important differences with the GCN architecture however. Firstly, the number of symbols is increased to allow for a measure of the proportion of the training patterns pertaining to each class addressing a given location. The number of *divisions* for each class is network dependent and will be defined so as to provide the necessary recognition performance.

Secondly, whereas GCN typically converges onto a single base symbol indicating the pattern class which has been identified, PCN typically converges onto a compound symbol indicating the relative probabilities of the sample pattern being a member of each pattern class. This is an important advantage of PCN as it allows the network to communicate that a sample pattern is, for example, most likely to belong to class x but is also quite close to class y whereas it is extremely unlikely to belong to class z. Typically of course the same pattern classes will appear to be similar and different due to the similar attributes possessed by the differing pattern classes (a '6' has more features in common with an '8' than a '7' for example). PCN does, however, have the capacity to give an indication of a degree of confidence which can be attached to its decision and this is a feature which may be important in a number of practical applications.

5.1 Performance of PCN

Example	Probabilities per class (thousandths)									
	0	1	2	3	4	5	6	7	8	9
1	250	69	49	57	96	132	142	54	85	6
2	265	66	62	44	117	86	94	78	79	109
3	288	72	61	62	82	91	142	43	92	67
4	332	45	61	75	71	100	112	43	89	72
5	248	65	57	46	102	119	99	81	92	91
6	253	55	74	61	97	115	78	81	85	101
7	145	87	72	49	174	69	120	95	84	106
8	262	62	74	56	114	86	105	63	87	91
9	294	51	69	72	71	104	80	62	94	104
10	275	56	78	81	86	93	79	69	85	98

Table 3: Examples of character '0' presented to PCN

The performance of the PCN architecture in a pattern classification task depends on the number of layers within each group, the connectivity patterns of the layers concerned and the number of divisions assigned to each pattern class. The increased complexity of the PCN architecture was designed primarily for employment on handwritten datasets. Some results obtained using a typical handwritten dataset presented with examples of the character `0' are displayed in table 1. For the given experiment, nine layers with varying connectivity patterns were employed per group with 1000 divisions per pattern class. There are ten pattern classes in total representing the numerals '0' through '9'.

The table displays the relative probabilities that the example pertains to each of the given classes. For example, example 1 is most likely to be a '0' and least likely to be a '2' with '0' overwhelmingly being most probable. Using the measure that the pattern class recognised is the class with the highest probability, the only incorrect recognition shown is example 7 where class '4' is most likely with '0' as second likeliest. Note however that the 'winning' probability is not high and that this recognition should be treated with some scepticism. Again using the measure that the class recognised is the one with the highest probability. overall performance for the entire class

of '0' characters was 92% and the average for all ten pattern classes was 82.1%.

6 Conclusions

A new paradigm for the construction of RAM-based neural network architectures has been presented which possesses the following properties:-

- The performance of a network may be adjusted by modifying the number of groups, layers within a group and the number of abstract symbols employed by the neurons within a given network.

- The ability to regularly modify the architecture implies that the simplest network possible meeting a given performance specification for an application may be employed.

- The existing benefits of RAM-based networks are retained in that the layers comprising the network require at most one shot learning and are trained independently.

- The architectures introduced allow the propagation of information across the network in an even and systematic manner and employ iterative convergence.

- The architectures employ multiple parallel layers.

References

1. G. Howells, M.C. Fairhurst, D.L. Bisset, BCN: A Novel Network Architecture for RAM-based Neurons, *Pattern Recognition Letters.*, Vol. 16. 297-303 March 1995.
2. G. Howells, M.C. Fairhurst, D.L. Bisset, `GCN: The Generalised Convergent Network', *Proceedings of Fifth IEE Intl. Conf. on Image Processing and its Applications*, Edinburgh, July 1995.
3. G. Howells, M.C. Fairhurst, D.L. Bisset, "PCN: The Probabilistic Convergent Network", *Proceedings of 1995 IEEE Intl. conf. on neural networks*, Perth, Western Australia, November 1995.

This page is intentionally left blank

Section 3

Applications of RAM based networks

For any approach to be worth while studying, demonstrable proof of its utility on practical problems is essential. This section contains a number of practical studies. All the applications are found in image processing, the traditional area for the successful use of RAM based methods (as in its original use). The main reason for this is that the methods scale well to the large input data sizes needed for image analysis problems. The final paper examines the implementation of ADAM, a RAM based network for image analysis, on a parallel system of transputers.

The first paper by O'Keefe and Austin shows there use in finding features in fax images. A problem that makes use of the potentially fast processing and noise tolerant properties as it is applied to faxes that are sent via typical fax machines. In addition it illustrates how RAM based methods compare with traditional object recognition methods.

Texture recognition is examined by Hepplewhite and Stonham, how introduce a novel pre-processing method and compare a number of existing N tuple pre-processing methods for this task.

RAM based networks are particularly suitable for small mobile robots as shown by Bishop, Keating and Mitchell, who demonstrate that a compound, insect like, eye can be created and used to control a simple robot.

Feature analysis is a vital part of machine vision explored by Clarkson and Ding. They show how a pRAM based network can be used to find features in a fingerprint recognition system. In addition, they show how noise injection can be used to improve performance of the method.

The use of colour in the detection of danger labels is investigated by Linneberg, Andersen, Jorgensen and Chistensen where the power of the N tuple method to solve real problems is demonstrated.

The problems involved in exploring complex images using saccadic image scanning methods is explored by Ntourntoufis and Stonham. They extend the MAGNUS network presented in section one to dealing with multiple objects in a 'Kitchen World' scene. Illustrating that iconic internal representations used in MAGNUS can be used to control image understanding systems.

Finally, hand written text is examined in the chapter by De Carvalho and Bisset, where the SOFT and GSN RAM based methods are combined in a modular approach to a difficult classification problem.

This page is intentionally left blank

CONTENT ANALYSIS OF DOCUMENT IMAGES USING THE ADAM ASSOCIATIVE MEMORY

S. E. M. O'KEEFE, J. AUSTIN

Advanced Computer Architecture Group, Department of Computer Science
University of York, York YO1 5DD, UK

An essential part of image analysis is the location and identification of objects within the image. Noise and clutter make this identification problematic, and the size of the image may present computational difficulties. To overcome these problems, a window onto the image is used to focus onto small areas. Conventionally, it is still necessary to know the size of the object to be searched for in order to select a window of the correct size. A method is described for object location and classification which allows the use of a small window to identify large objects in the image. The window focusses on features in the image, and an associative memory recalls evidence for objects from these features, avoiding the necessity of knowing the dimensions of the objects to be detected.

1 Introduction

This chapter describes the application of the ADAM associative memory to a specific image analysis problem, that of identifying arbitrary objects in an image. In particular, the problem domain is that of document images, and the broad objective of the analysis is the classification of the image according to the document's content. The bulk of the content of the document will be found in text, and there has been much work on the segmentation and processing of printed text. However, the make-up of a document is not limited to printed text—the non-textual objects in the document image may provide important contextual information to guide the analysis of the rest of the document.

Each object is modelled as a number of features, not necessarily unique, in a particular fixed spatial relationships to each other. When analysing an image, if the features which comprise the object are correctly identified, and if they appear in the image in the correct relationship to each other, then it is possible to determine the presence of the object in the image to a level of confidence based on the certainty of correctly identifying the object features. This is the basis of the Generalised Hough Transform (GHT) (Ballard[1]), used to detect arbitrary shapes in images. The work described here uses the ADAM associative memory neural network to implement the GHT.

2 Generalised Hough Transform

The Generalised Hough Transform is an extension of the Hough Transform [2,3] to permit the detection of arbitrary, non-analytic shapes. The Hough transform is summarised briefly below, followed by a description of the Generalised Hough Transform.

The original Hough Transform is a method for detecting curves by exploiting the duality between points on a curve and the parameters of that curve. The points which make up the image of a curve are positioned by the parameters which describe the curve – if the parameters of the curve are changed, the positions of the points making up the curve are changed. Conversely, if the points of the curve are shifted, then the parameters required to describe the curve are changed. Given this relationship between the points on a curve and the parameters describing the curve, then each point can be mapped into a set of possible parameter values representing the family of curves on which the point may lie. Taking each point in turn, the sets of possible parameter values may be accumulated. The intersection of these sets represents the set of parameters common to all the points, and thus the parameters which describe the curve.

In the case of the Generalised Hough Transform, the relationship between features in the image and the objects which they represent is recorded in a look-up table, linking each feature with a set of parameter values. Typically, the parameters considered will be the position, orientation and scale of the object. Rather than individual pixels, the features extracted from the image may be edge elements comprising the boundary of the object. In the work presented here, the features used are blocks of pixels representing arbitrary features of the object, and the parameterisation is of feature position and class of object – for each feature, the look-up table stores the class of object in which it occurs and the position of the feature relative to an arbitrary reference point on the object. When a feature is detected, the position information is recalled and combined with the actual position of the feature to give a position for the object. A label for this class of object is accumulated at this position. This parameterisation is therefore translation-invariant. Scale and rotation invariances are not considered.

Object detection is achieved by location of maxima in the accumulator. Typically, the GHT is used to search for a single class of object, and the maxima in the accumulator indicates the particular parameters of the dominant instance of that object. In the implementation described in Section 4, multiple classes of object may be searched for in parallel, and each instance of each class of object will be represented by a peak in the accumulator.

The accumulation of evidence for an object depends upon the location and identification of the features which comprise the object. If the object in the image does not match the object template perfectly because of noise, clutter or other distortions, then the probability of detecting and correctly identifying each feature is reduced, and the probability of identifying the entire object is therefore also reduced. The accumulation of discrete pieces of evidence requires the quantisation of the parameters spanning the accumulator. This quantisation makes the algorithm sensitive to small changes in features and their measurement. Thus, noise and clutter may effect features, which leads to the dispersion of evidence throughout the accumulator rather than the formation of a discrete peak. Grimson and Huttenlocher [4] have analysed the Generalised Hough Transform in detail and produced a formal model of the errors introduced by the quantisation and measurement processes. They model the effect of inaccuracies in measurement, and predict the effect of these errors on the accumulation of evidence. The results of their analysis show the method to be sensitive to the number of features which comprise an object model. Using a simple model of noise and clutter in the image, they also show the method to be prone to the generation of false positives.

In the GHT, then, the object model is in the form of a template which describes the features comprising the object, and their relationship to each other. Matching a model to the image data involves a search through all available models to decide which model best fits the data. This search may present a computational problem when there are a large number of models. This is the case when analysing real images. The computational problem is solved by implementing the Generalised Hough Transform using a fast associative memory, ADAM (Advanced Distributed Associative Memory)(Austin and Stonham [5], O'Keefe and Austin [6,7]). ADAM is a neural network specifically designed for image analysis tasks (Austin et al. [8]). Put simply, the network allows for the association of input and output patterns in a compact form, so that the presentation of an input pattern stimulates the output of the associated pattern without a serial search of the pattern memory. The network has binary weights, and therefore may be held in memory in a compact form, with only one bit per weight. Thus, the network is ideal for performing the sorts of template matching involved in the GHT.

The network is also able to generalise about the features it has been trained to recognise, so that it is able to recognise noisy or corrupted versions of them. It can return a measure of confidence in the recognition of each feature, and can thus be made robust to corruption of the image by noise at the feature level.

3 The ADAM neural network

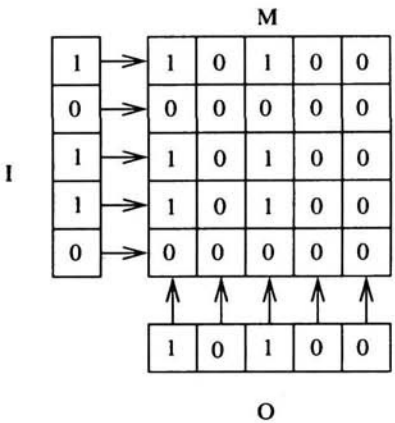


Figure 1: Binary correlation matrix M, with input A and output B

ADAM is a fast, high-capacity associative memory, designed for use in image analysis tasks. The layout of ADAM is shown in figure 2. ADAM uses binary correlation matrix memories (CMMs) to learn and recall associations between input and output arrays of binary data. An example of such a matrix is shown in figure 1. A is the input array and B is the output array. The matrix M represents the outer product of A and B. All elements of M are set to 0 initially. During training, the elements of M record the intersections of the rows corresponding to the bits set to 1 in A, with the columns corresponding to the bits set to 1 in B. At these intersections the elements of the arrays are set to 1. Each subsequent pattern is superimposed on existing patterns, forming the logical OR of the patterns.

To recall an associated pattern from the memory, the test pattern is placed in A and an inner (or dot) product is formed, with the result placed in B. The result will be an integer array which must be thresholded to give a binary result. The thresholding method used is that developed by Austin [5], and is referred to as N-point thresholding (sometimes termed L-max, Casasent and Telfer [9]). In this method, every original pattern in B has N bits set. On recall, the N elements of the array which have the highest value are set to 1, and all others are set to 0.

The basic structure of ADAM is shown in figure 2. ADAM makes use of two CMMs. Because the input and output patterns may be very large, a

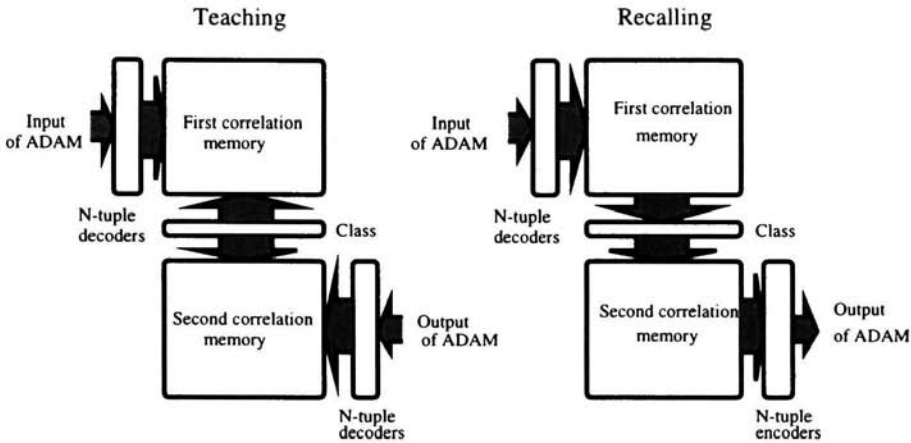


Figure 2: The ADAM associative memory

single CMM associating input and output would be impractical. ADAM uses an intermediate code or "class" which is much smaller than either of the input or output arrays. The first CMM learns associations between the contents of the input array and some intermediate code, selected to be unique for that input, and the second CMM learns the association between the intermediate code and the output. In this way, the total size of the matrices is reduced.

The input array is tupled. This preprocessing of the input splits it into groups of binary elements ("tuples"). Each tuple decoder interprets the values in its tuples as representing a state. The decoder has a separate output for each state, and sets one, and only one, output corresponding to the state of the tuple. As a consequence of this tupling of the input, linearly inseparable patterns may be classified; the input to the first CMM is much more sparse than the original data, thereby reducing saturation of the memory and increasing capacity; and the number of bits set in the input to the CMM is fixed, making thresholding of the output from the CMM simpler. The properties of the memory can be controlled by selecting the size of tuples in the input and output, the size of the intermediate class, and the number of bits set in the class.

4 Implementation

An object recognition tool for use in the analysis of images has been implemented, using the ADAM neural network to implement the GHT. A block diagram of the structure of the recogniser is shown in figure 3. The recogniser

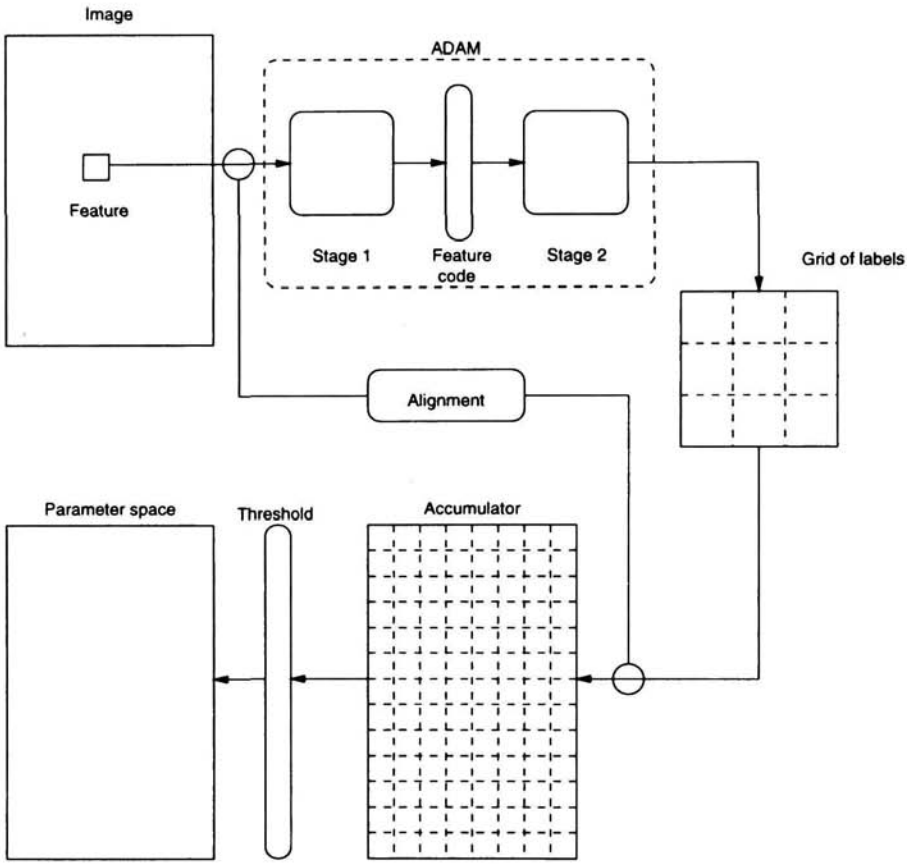


Figure 3: Implementation of the recogniser

is trained to associate object features with a parameterised description of the object. The recogniser extracts a feature from every point in the image. For features that it has been trained to recognise, the associative memory recalls the associated parameter values. In this implementation, the parameter values refer to the object class and its position relative to the feature position. The evidence for the existence of the object in the image is accumulated as the image is searched. When feature have been extracted from the whole image, the accumulated evidence at each point in parameter space is analysed to determine the location and class of objects in the image.

The object is modelled as a set of associations. Each association is between a feature and one or more vector/label pairs. The label is a tag for the class of object, and the vector is the position of the notional object "centre" relative to the feature. The features which are used to describe the objects are $m \times m$ blocks of pixels selected from a training example of the object.

The label used for the class of the object is an N-point code. That is, it is a string of binary elements of which exactly N are set to 1. This method of encoding the object class allows different labels to be superimposed when evidence is accumulated, and allows the dominant label at each position to be retrieved reliably.

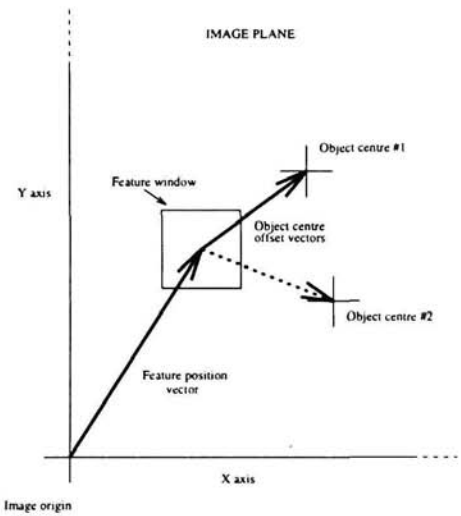


Figure 4: Calculation of object position from feature position and recalled parameters

The position vector is encoded by approximating it to a position in a grid of points surrounding the feature position. Each position in the grid contains the labels for object classes which have appeared at this position relative to the feature during training. The grid is small compared to the size of the image. For example, it might be a 15×15 point grid, covering an area of 75×75 pixels around the feature. The feature is associated with this grid data structure.

As with the GHT, each feature may occur a number of times, and in many different classes of objects. The feature is therefore associated with a number of vector/label pairs. The template for the object search is the set of all the features and their associated parameter values.

To locate an object in an image, the recogniser must search for the $m \times m$ blocks of pixels which constitute the features which have been learned. When a feature is found in the image, the corresponding vector/label pairs are found from the template. From the position of the feature, and the recalled position of the object centre, the location of the putative object in the image is calculated (figure 4). At this position, the label for the object is accumulated. This is repeated for all vector/label pairs that has been recalled for the feature, and this is repeated for every feature found in the image.

When the whole image has been scanned for features, the accumulated evidence is examined. When an object is actually present in the image, and all or most of its features are found, a large number of labels are accumulated for that object at the position of the object centre. This is apparent as a peak in accumulated evidence. Each object recognised is represente by a peak, and there will also be peaks due to clutter and noise in the image.

The recogniser is trained by teaching the ADAM to associate features ($m \times m$ blocks of pixels) with object labels and positions. Once the ADAM has been trained, the recogniser can search for the object in any image. At every position, an $m \times m$ block of pixels is extracted. If the ADAM recognises this block of pixels as a known feature, it recalls the associated grid structure containing the vector/label pairs. Recognition of a feature is determined by the confidence with which the first CMM produces the intermediate code — a high confidence implies recognition of a feature, and low confidence implies the feature is not recognised. Object labels appear on the grid points corresponding to the object position relative to the feature. This may be performed in parallel by an array of ADAM networks, although it is currently simulated in software.

The contents of this recalled grid are added into an accumulator grid. The accumulator grid covers the whole image, with cells in the accumulator grid mapping onto points in the image. The grid structure associated with a feature, and recalled from the memory, is added into the accumulator at the position of the feature in the image. Thus, at each position in the accumulator some number of labels is accumulated, corresponding to putative objects in the image, for which there is some feature evidence belonging to the object.

Once the whole image has been scanned for features, and the evidence has been accumulated, object positions are determined. This is done by looking at every cell in the accumulator and determining whether the accumulated evidence in that cell exceeds the threshold for an object. Assuming that the ADAM has been trained to recognise more than one object, then each cell of the accumulator may hold labels for more than one class of object. This is illustrated in figure 5. To determine which is the dominant class of object at this location, the contents of the accumulator cell is N-point thresholded. That

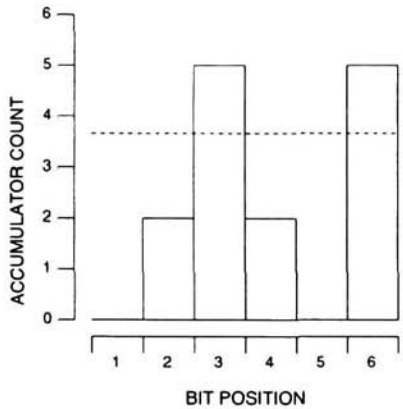


Figure 5: Labels for more than one class of object accumulated in one cell

is, the N largest elements in the cell are selected. This gives us the object label "0 0 1 0 0 1". To determine the level of confidence in this label, the difference between the "strength" of this label and the other labels in the cell (which are treated as noise) is compared with the response expected from the object model.

5 Results

To demonstrate the capabilities of the recogniser, it has been applied to the problem of recognising objects in fax images. These images present difficulties because of their size, and the presence of noise in the images.

The test images are shown in figure 6. The clean data is in figure 6(a). The characters in the test image are shown at their actual size, and are treated as arbitrary objects to be identified, rather than as characters *per se*. The recogniser has been trained on one instance of the object "A" in the fragment of image shown in figure 6(a). The feature size is 11×11 pixels, and the position of the feature relative to the object is approximated by an 19×19 grid, with 10 pixels between grid points. The recogniser was then presented with both the clean image used for training, and a corrupted version of the same image (figure 6(b)). The recognition process takes in the order of one minute, running on a Hewlett Packard HP9000/715 workstation (33MHz).

Figure 7 shows the results of the processing. The x and y axes of the plots correspond to the x and y axes of the images. The z axis shows the confidence with which the class of objects has been located at each point in the image.

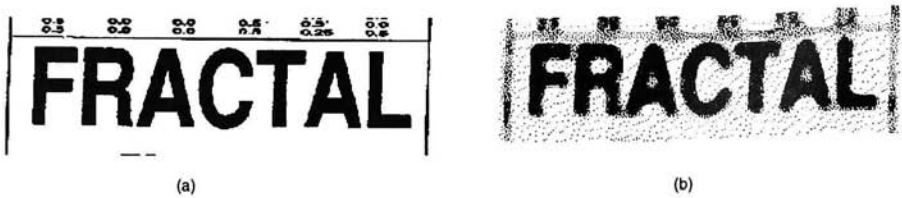


Figure 6: Test images (a) fragment of a fax (b) fragment after addition of noise

The higher the peak on the map, the higher the confidence that the object is at that point (confidences are scaled to a maximum of 256). It can be seen in figure 7(a) that both instances of object "A" have produced peaks in the accumulator. However, the detection of the objects is far from satisfactory. The left-hand peak in figure 7(a) is actually a double peak, indicating that the evidence for the object has been spread over two adjacent cells in the accumulator. This is due to the coarseness of the quantisation of the vector. The right-hand peak is lower than the left, showing a lower confidence in the detection of this object. There is also a considerable amount of noise in the accumulator. This is the result of detection and recognition of features which are not part of an object.

Figure 7(b) shows the large reduction in confidence which occurs when noise is added to the image. Both objects are barely distinguishable.

The presence of clutter (extraneous features) and noise complicates the recognition process. In order to overcome these problems, changes to the parameterisation used to model objects and the introduction of feedback into the recognition process have been implemented [11]. The object will be modelled in terms of the position of each features relative to all others within a certain distance. Recognition of a feature will result in the recall of a grid structure containing labels for features, as well as labels for objects. The feature information will be fed back to the recogniser, so the recognition process at any position will be informed by the image data at that position, and the supporting evidence from nearby features. Where there is insufficient supporting evidence, the feature will not be recognised. In this way, clutter will be suppressed, and noisy features which are part of an object are more likely to be recognised, improving the recognition confidence.

Another step in the development is the transition to specialised hardware. The C-NNAP processor (Austin et al. [10], and chapter 3.9) is designed to implement ADAM directly in hardware, and to run ADAMs in parallel. Migration to this hardware will give two orders of magnitude increase in the speed with which images are analysed.

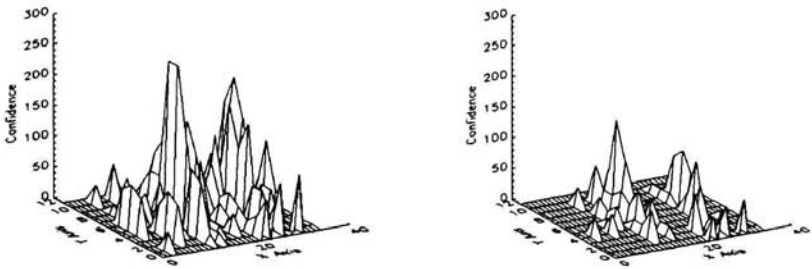


Figure 7: Results of processing. Each map shows the confidence with which the class of objects has been located at each point in the image. Confidences are scaled to a maximum of 256.

6 Conclusions

We have implemented an object recogniser, using a GHT-like algorithm. The ADAM has been used to provide a fast associative look-up. The drawbacks of the system at the moment relate to the cost of implementing ADAM in software. Even for small numbers of object models with relatively few features, large correlation matrix memories are needed, so the benefits of this implementation are not apparent for small problems. The system also suffers from noise and clutter in the image, and the next stage of development is the inclusion of feedback, so that the recogniser can iterate towards a solution, labelling objects confidently and suppressing the effects of noise.

However, the system can search for multiple objects in parallel — features in the image prompt the recall of the associated object or objects directly, without a serial search of the memory. The search of the image for features is also an obvious candidate for parallelisation, since the detection of a feature and the recall of the object information at any location is independent of feature detection in the rest of the image. When the recogniser is implemented using the C-NNAP hardware, rather than simulating this in software, a considerable increase in speed is expected.

References

1. D. H. Ballard. Generalising the Hough transform to detect arbitrary

shapes. *Pattern Recognition*, 12:111–122, 1981.
2. R. O. Duda and P. E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–, January 1972.
3. J Illingworth and L Kittler. A survey of the hough transform. *Computer Vision, Graphics and Image Processing*, 44(1):87–116, 1988.
4. W.E.L. Grimson and D.P. Huttenlocher. On the sensitivity of the Hough transform for object recognition. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 12(3):255–274, 190.
5. J. Austin and T.J. Stonham. Distributed associative memory for use in scene analysis. *Image and Vision Computing*, 5(4):251–260, Nov. 1987.
6. S. O'Keefe and J. Austin. Application of an associative memory to the analysis of document fax images. In E. R. Hancock, editor, *Proceeding of the British Machine Vision Conference*, volume 1, pages 315–326. British Machine Vision Association, BMVA Press, 1994.
7. S. O'Keefe and J. Austin. Image object labelling and classification using and associative memory. In *IEE Fifth International Conference on Image Processing and its Applications*, pages 286–290. Institution of Electrical Engineers, London, July 1995.
8. J Austin, M Brown, I Kelly, and S Buckle. ADAM neural networks for parallel vision. In *JFIT Technical Conference 1993*, pages 173–180, 1993.
9. D. Casasent and B. Telfer. High capacity pattern recognition associative processors. *Neural Networks*, 4(5):687–698, 1992.
10. J Austin, J Kennedy, S Buckle, A Moulds, and R Pack. The cellular neural network associative processor, C-NNAP. In *IEEE Monograph on Associative Computers.*, 1997.
11. S. E. M. O'Keefe. *Neural-based content analysis of document images.* PhD thesis, Department of Computer Science, University of York, 1997.

TEXTURE IMAGE CLASSIFICATION USING N-TUPLE CODING OF THE ZERO-CROSSING SKETCH

L.HEPPLEWHITE, T.J.STONHAM
Neural Networks and Pattern Recognition Research Group,
Department of Electronics and Electrical Engineering,
Brunel University, Uxbridge, Middlesex,
UB8 3PH, UK

A novel approach to real time texture recognition, derived from the n-tuple method of Bledsoe and Browning, is presented for use in industrial applications. A wealth of texture recognition methods are currently available, however few have the computational tractability needed in an automated environment. Methods based on the nth order co-occurrence spectrum are discussed together with their shortcomings before a new method which uses nth order co-occurrence methods to describe texture edge maps instead of pixel intensity values is presented. The resulting co-occurrence representation of the texture can be classified by established statistical methods or weightless neural networks. Finally the new method is applied to the problems of texture classification and segmentation.

1 Introduction

Texture classification remains a fundamental task for image processing. In recent years, various approaches have been presented for the solution of the texture classification problem. [1-2] The study of texture has been greatly influenced by theoretical considerations, particularly the early work of Julesz [3], which concentrated on the statistical properties of textures which can be segregated pre-attentively. This phenomenon is illustrated in fig. 1b where the X-shaped patterns appear unconsciously against a background of L-shaped patterns. However, the rectangle of T-shaped patterns (rhs of fig. 1b) is difficult to see and requires close scrutiny.

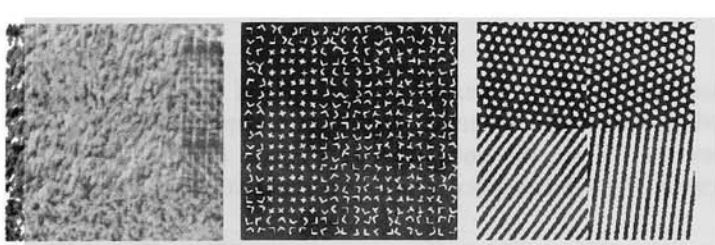


Figure 1: Texture segregation a.Brodatz collage b.Julesz Texture c.sinusoidal gratings

Subsequent methods of texture classification have been based on the statistical properties of the texture regions. Similarly, properties of psychophysically identified spatial mechanisms gave rise to methods characterising texture regions by orientation and spatial-frequency content.(fig. 1b) However, few of these methods have the computational tractability needed in industry applications. N-tuple pattern recognition methods, derived from the work of Bledsoe and Browning [4], have achieved some success in real-time pattern recognition tasks [5] and were applied to texture recognition by Patel and Stonham [6] in the Texture Co-occurrence Spectrum of n^{th} order.

2 Texture Co-occurrence Spectrum of n^{th} order

Common to many texture recognition methods, inspired by the work of Julesz, are the concepts of micro- and macro-texture. Micro-texture can be viewed as the description of the texture's primitive elements, or Textons in Julesz terms, whilst macro-texture is the distribution of these primitive elements. In the TCS method, micro-texture information is extracted from an image, $P(x, y)$, in n pixel samples, termed n-tuples, using oriented local neighbourhood operators, m_θ.

$$m_\theta(i) = P\left(x + k\left(i - \frac{n}{2}\right) cos\theta, y + k\left(i - \frac{n}{2}\right) sin\theta\right) \qquad i = 0..n \qquad (1)$$

Hence n-tuple samples extracted from a given texture can reflect micro-texture information of various spatial frequencies, spatial extent and orientation by suitable selection of the tuning parameters: tuple size, n, interpixel spacing, k, and orientation, θ. Each oriented n-tuple sample taken from an image containing g intensity levels corresponds to a n-tuple state, $S_\theta \in [0, g^n - 1]$, from the universe of possible states assigned as follows:

$$S_\theta = \sum_{i=0}^{n} g^i . m_\theta(i) \qquad (2)$$

Subsequently a macro-texture description of the texture can be obtained by recording the relative occurrence of these n-tuple states, S_θ, within a region of the texture. Thus, discontinuities in texture (see fig. 1a) are reflected by differences in the relative occurrence of n-tuple states.

2.1 *Binary Texture Co-occurrence Spectrum (BTCS)*

In application to binary texture recognition (i.e. $g = 2$) the method characterises a sample texture by recording the occurrence of these n-tuple states over the texture sample to form a 2^n dimensional state vector. These state vectors, termed co-occurrence spectra, can be readily sorted by established pattern recognition techniques including weightless neural networks. The simplicity and computational efficiency of this method make it an attractive solution to real-time texture classification. However real world textures are rarely binary in nature and the method's performance is restricted by the effectiveness of the thresholding techniques employed.

2.2 *Grey Level Texture Co-occurrence Spectrum (GLTCS)*

Unfortunately the BTCS does not scale up to the multivalued grey level case since a grey level image containing g intensity levels would produce a co-occurrence spectrum of dimensionality g^n. This necessitates some approximation of the n-tuple states to be made in order to reduce the dimensionality. Typically Rank coding is utilised [7–9] reducing the dimensionality to $n!$ Again the method appears attractive to the industrial inspection environment as demonstrated by the authors. [10]

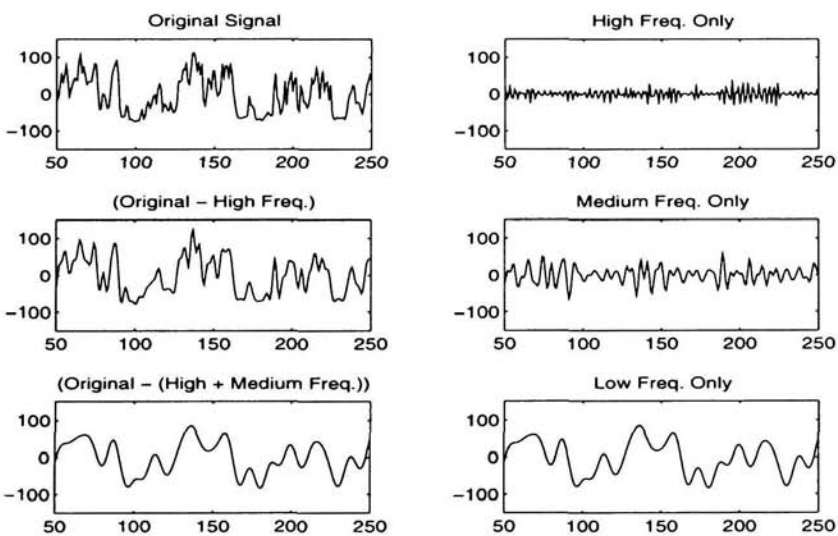


Figure 2: Intensity profile taken from Tree Bark Texture and it's Pass-Band decomposition

However in reducing the dimensionality needed to represent the occurrence of the n-tuple states, rank coding is indiscriminant in the way it divides the pattern space; little or no attention is given to what textural features the rank codes are extracting. Since n-tuples are extracted by local neighbourhood operators with a particular interpixel spacing, they represent intensity changes at a certain spatial frequency within the texture. Optimal selection of the interpixel spacing, k, remains ad hoc; however the Nyquist sampling theorem sets a lower bound on k relating to the highest spatial frequency present within the texture.

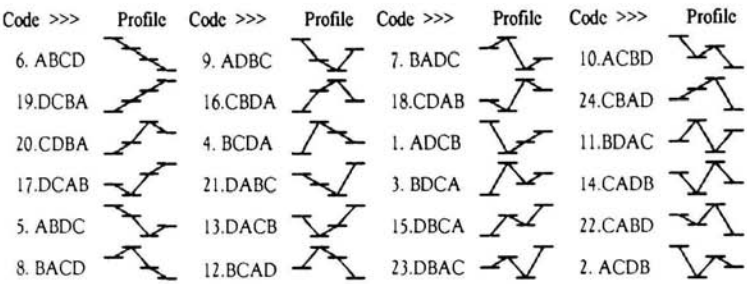


Figure 3: Rank coding of 4-tuples and the luminance profiles they represent

Experimentally high frequency information is also found to improve texture discrimination. However, texture is multi-resolution in nature and thus contains intensity changes at various spatial frequencies, as demonstrated by the tree bark intensity profile of fig. 2. Clearly, n-tuples extracted at various sampling frequencies will also contain intensity changes due to lower spatial frequencies.(fig. 2 lhs) The distribution of energy within real world images decreases exponentially with increasing frequency and thus intensity variations decrease accordingly. (fig. 2 rhs) Hence low frequency variations dominate and n-tuples extract low order intensity profiles, typically 'edge' profiles. Unfortunately, as fig. 3 demonstrates, rank coding biases the coding towards higher order profiles which are less common in the data and more susceptible to noise.

Hence this results in the co-occurrence spectrum appearing similar for all textures. Wang and He [11] present a similar method, the Texture Unit and Texture Spectrum (TUTS), which attempts to code local 3x3 window profiles as texture units. Examination of the texture spectra produced by this method confirms that only a discrete set of the 6561 possible texture units occur regularly in the textures. Again these correspond to 'edge' profiles at various orientations.

The TCS methods can sample at these lower spatial frequencies through the interpixel spacing, k, and tuple size, n. However, as demonstrated lower spatial frequencies remain in each n-tuple sample biasing the high frequency samples and making parameter selection difficult. In short, intensity changes can not be represented at all scales simultaneously and the TCS method must be 'tuned' to a particular scale. (see fig. 2 rhs)

3 Zero Crossings Texture Co-occurrence Spectrum (ZCTCS)

In order to detect intensity changes at different scales an appropriate filter must have two salient characteristics: it should be a gradient operator and be capable of tuning to a particular scale. Marr and Hildreth [12] propose the Laplacian of Gaussian filter (eqn. 3) as a computationally optimal solution. This LoG filter tunes to a particular scale of representation by spatially blurring the texture (i.e. low pass filtering it). This blurring is performed by the Gaussian operator which has optimal localisation in both spatial and spatial frequency domains. Hence the Gaussian performs an optimal trade off between band limiting the frequency content of the texture and preserving spatial features of the texture (e.g. edges).

$$\nabla^2 G(x,y) = \left(1 - \frac{x^2+y^2}{2\sigma^2}\right) e^{\frac{-(x^2+y^2)}{2\sigma^2}} \tag{3}$$

The second characteristic of the LoG filter, the gradient operator, is performed by the Laplacian operator. This performs the second derivative of the band limited texture thus representing edges at this scale as zero-crossings in the resulting filter output.(see fig. 4) Since orientational features are extracted by the subsequent n-tuple operators, the Laplacian is chosen as the lowest order isotropic derivative operator. An important property of the resulting LoG filter is it's balance between positive and negative values. This means that the response to dark/light edges is exactly opposite to that of light/dark edges. The image resulting from convolution of this filter with a texture defines edges within the texture as zero crossings in the new image.(see fig. 5)

Figure 4: Intensity change in image gives rise to a peak in the first derivative and a zero crossing in the second derivative

Logan's Theorem[13] indicates that a band pass signal can be reconstructed from its zero crossings under certain circumstances. Although this theory is not strong enough to make any similar claims about texture features, it is clear that the zero crossings are rich in information. Hence, in order to reduce the complexity of future processing the output of the filter is binarized. This binarization acts as a 'blob' detector whilst preserving the sign of the zero crossings present in the image. As can be seen in fig. 5 the grey level texture recognition problem has been reduced to a binary texture recognition problem which can be resolved by the BTCS. The resulting method retains the attractiveness of the BTCS whilst making it independent of threshold level at the minor computational expense of a simple pre-processing operation.

3.1 Parameter Selection

Since the *LoG* filter can be closely approximated by the difference of two Gaussians[12] with a ratio of $\frac{\sigma_{lower}}{\sigma_{upper}} = 1.6$, the filter output is band limited with upper and lower half-height cut-off frequencies given by:

$$f_{upper} = \frac{\sqrt{2ln2}}{2\pi\sigma_{upper}} \tag{4}$$

$$f_{lower} = \frac{\sqrt{2ln2}}{2\pi\sigma_{lower}} = \frac{f_{upper}}{1.6} \tag{5}$$

The BTCS can now sample the filter output recording the co-occurrence of zero crossings. Hence the optimal interpixel spacing, k, is given by the Nyquist sampling theorem:

$$k \leq \frac{1}{2f_{upper}} = \frac{1}{3.2f_{lower}} \tag{6}$$

Consequently the n-tuple size, n, can be defined by the maximum number of zero crossings a single n-tuple is to represent, z_{max}, and the lowest frequency at which they occur, f_{lower}.

$$n \approx \frac{z_{max}}{2kf_{lower}} \tag{7}$$

Hence in a controlled industrial environment, where a small discrete set of scales will be important, the dominant scale can be chosen by setting the centre frequency of the *LoG* filter. This automatically sets the above parameters of the new texture method termed the Zero Crossing Texture Co-occurrence Spectrum (ZCTCS).

4 Texture Classification

In order to demonstrate the effectiveness of the new method in texture classification, it's performance has been compared with the BTCS, GLTCS and the TUTS of Wang and He. [11] To evaluate each method's discriminating performance, classification has been performed using textures taken from the Brodatz Texture Album. [14] The 30 textures used (see fig. 5 for a subset) were chosen to represent finer scale textures, with the exception of textures d5, d28 and d85, allowing the LoG of the ZCTCS to be tuned to the dominant fine scale ($\sigma_{upper} = 1pixel$). Twenty non-overlapping sample windows ($32 \times 32pixels$) of each texture were classified using a *leave-one-out* procedure and a simple Euclidean distance classifier. [15]

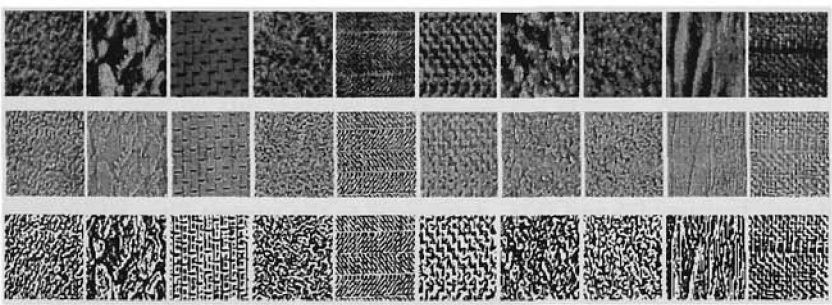


Figure 5: Brodatz textures (top row), LoG filtered (middle), Binarized LoG (bottom) Textures(L to R): pressed cork(d4), expanded mica ×3(d5), woven aluminium wire(d6), grass(d9), herringbone weave(d16), herringbone weave ×4(d17), beach sand ×4(d28), beach sand(d29), water(d38), straw matting(d85)

Table 1 shows the classification results for various interpixel spacings. A window size of 32 pixels was chosen as it effectively extracts typical archetypes for the textures. The texture micrographs were $384 \times 384pixels \times 256grey\ level$ images taken from the Brodatz texture album [14]. In the BTCS, GLTCS and the ZCTCS methods, four n-tuple operators, $\theta = 0^o, 45^o, 90^o, 135^o$, were used to extract textural information from 4 orientations and form 64, 96 and 64 dimensional texture spectrums respectively. To binarize the textures a global threshold level of 80 was used in the BTCS method. The TUTS [11] method was implemented using a 3×3 operator and a 6561 dimensional texture spectrum. The ZCTCS method was implemented with $\sigma_{upper} = 1pixel$ which subsequently defines the interpixel spacing, $k = 2pixels$, and the n-tuple size, $n = 4$, for pairs of edges.

Table 1: Results of classification

	$k=1$	$k=2$	$k=3$	dimen.	miss-classified textures
BTCS	79 %	-	-	4×16	d5,d6,d28,d29,d38
TUTS	97 %	-	-	6561	d17,d28,d29
GLTCS	98 %	97 %	93 %	4×24	d28,d29,d38
ZCTCS	100 %	100 %	97 %	4×16	d5,d28
ZCTCS†	95 %	100 %	92 %	16	d5,d28

† *denotes cross operator*

Also shown in the results is the performance of the ZCTCS with a simple cross operator instead of the four orientational masks. As the results demonstrate the performance of the ZCTCS is comparable with others methods. However the important advantage of the method is in which samples are miss-classified. Any misclassifications made by the ZCTCS are exclusively associated with textures D5 and D28. Analogous to the current filter which discriminates the perceptually similar d4, d9 and d29, another filter would be needed for the d5 and d28 pair.

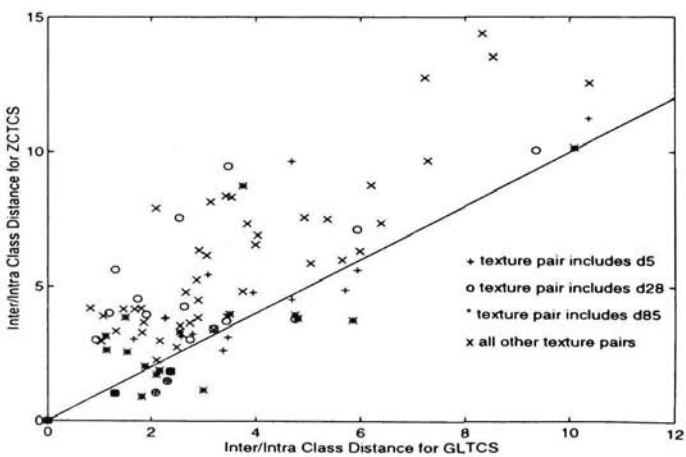


Figure 6: Comparison of Inter/Intra Class Separation for ZCTCS and GLTCS

Investigation of each of the above method's feature space indicates that while the intra-class scatter remains constant across methods the ZCTCS clearly increases the inter-class separation of those textures for which it is *tuned*. Thus as demonstrated in Table 1, this increase in inter-class separation improves the overall discrimination of the ZCTCS method. Figure 6 demonstrates the ZCTCS's improvement in class separation over the GLTCS method. Clearly the only texture pairs for which the overall class separation is lowered, relative to the GLTCS method, are those containing textures d5, d28 or d85. In other words, those for which the ZCTCS is not optimally *tuned*.

5 Texture Segmentation

A common industry application for texture recognition methods is that of supervised texture segmentation. This involves scanning a moving window across an image and assigning the underlying region of the image to a previously defined class based on the textural content of that region. Assignment of each window is analogous to the above classification of texture samples. Hence, the increased discrimination of the ZCTCS method should result in improved segmentation.

Figure 7: Texture collage segmentation results using GLTCS and ZCTCS methods

This is demonstrated in Fig. 7 where the segmentation performance of the ZCTCS [16] is contrasted with that of the GLTCS. Each method was previously trained using 20 samples of the 30 Brodatz textures used in the classification experiments. The test image was segmented by stepping a 32×32 *pixel* window across the image in 4 *pixel* steps. The underlying region was assigned as one of the 30 textures using a Euclidean distance classifier.

Figure 7 shows the resulting segmentation for a non-trivial collage of the perceptually similar d4, d9, d29 together with d85. Clearly the ZCTCS outperforms the GLTCS in segmenting the difficult d4, d9, d29 collage at the expense of the d85 boundary to which the LoG is not optimally tuned. It should also be noted that, in practice, both segmentation results would be improved by post-processing with, for example, a median filter. However, the raw segmentation results again confirm the improved discrimination performance of the ZCTCS method predicted by the feature space analysis of section 4.

6 Discussion

The new Zero Crossing Texture Co-occurrence Spectrum method has been shown to extend the usefulness of n-tuple pattern recognition methods for texture recognition. Results presented for texture classification and texture segmentation demonstrate how the n-tuple methods can now be *tuned* to certain spatial frequencies in the texture. By tuning the n-tuple methods to spatial frequency, the ZCTCS also addresses the previously ad hoc selection of parameters. The n-tuple size, n, and interpixel spacing, k, have been shown to be related to the spatial frequency to which the Laplacian of Gaussian is tuned.

Due to the low dimensionality of the method's representation, particularly the cross operator, several spatial frequency bands could be described at once without degrading performance significantly. In particular, by approximating the Laplacian of Gaussian by a Difference of Gaussians a computationally efficient filter bank can be created using the X-Y separability of the Gaussian. The ZCTCS thus retains the computational efficiency and simplicity of the binary n-tuple methods making real-time texture segmentation possible.

Acknowledgments

The authors acknowledge funding of this research by Xyratex formerly IBM Havant U.K.

References

1. L. Van Gool, P.Dewaele and A.Oosterlinck, *Comp. Vision, Graphics and Image Proc.* **29**, 336 (1983).
2. M. Tuceryan and A.K. Jain, *Handbook of Pattern Recognition and Computer Vision*, (World Scientific, 1993).
3. B. Julesz, *Nature* **290**, 91 (1981).
4. W.W. Bledsoe and I. Browning, *Proc. of the Eastern Joint Comp. Conf.* , 225 (1959).
5. I. Aleksander and T.J. Stonham, *Computers and Digital Tech.* **2**, 29 (1979).
6. D. Patel and T.J. Stonham, *Proc. IEEE Int. Symp. on Circuits and Systems* , 2657 (1991).
7. J. Austin, *Proc. 4th Int. Conf. on Pattern Recognition* , 28 (1988).
8. D. Patel and T.J.Stonham, *Proc. SPIE Visual Comms. and Image Processing* **1818**, 1206 (1992).
9. L. Hepplewhite and T.J. Stonham, *Proc. 12th Int. Conf. on Pattern Recognition* **A**, 589 (1994).
10. L. Hepplewhite and T.J. Stonham, *Image Processing* **7**, 24 (1995).
11. L. Wang and D.C. He, *Pattern Recognition* **23**, 905 (1990).
12. D. Marr, *Vision*, (W.H. Freeman & Co., 1982).
13. B.F. Logan, *Bell Syst. Tech. J.* **56**, 487 (1977).
14. P. Brodatz, *Texture: A Photographic Album for Artists and Designers*, (Dover, 1966).
15. R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis*, (Wiley, NY, 1973).
16. L. Hepplewhite and T.J. Stonham, *Electronics Letters* **33**, (1997)

A COMPOUND EYE FOR A SIMPLE ROBOTIC INSECT

J.M.BISHOP, D.A.KEATING, R.J.MITCHELL
Department of Cybernetics, The University of Reading,
PO Box 219, Whiteknights, Reading, RG6 6AY, UK

The Department of Cybernetics has recently developed some simple robot insects which can move around an environment they perceive through simple sensors. Suitable sensors currently implemented include proximity switches, active and passive infra red detectors, ultrasonics and a simple compound eye. This chapter describes how a such an eye linked to a simple weightless neural network can be used to give an estimate of position within a complex environment. Such information could be used by the insect to generate more intricate behaviours.

1 Introduction

There is much interest in the development of intelligent machines which can learn from their environment. Various machines have been developed which have many sensors, sometimes including high resolution video, which require a great deal of computing power to process the information coming in to the machine. More processing is then required to determine suitable action in response to this information. Researchers in the Department of Cybernetics at the University of Reading believe that it is best to start with much simpler systems. Also, we believe that there is much to learn from the behaviour of simple organisms like insects. Therefore, a number of simple robotic insects have been developed which are small and which can operate rapidly[1]. The first systems to be built have few sensors which the insects use to determine a limited (though not trivial) behaviour. However, the insects were designed so that extra sensors could be added to allow more complex behaviour.

The simple insects have two ultrasonic sensors, which enable the insect to detect how far the nearest object is in front of a sensor, and two motors, each of which can be set to move forwards or backwards at a given speed. The actions of the insect are determined using a simple weightless network pre-programmed into an EEROM. A binary pattern, corresponding to the data from the ultrasonic sensors, is passed to the address lines of the EEROM, and the value at the addressed location specifies the speed and direction of each motor. Thus the operation of network defines one set of simple insect behaviours. Different behaviours can be selected by using DIL switches, providing extra address lines to the EEROM.

The ultrasonic sensors are controlled by a programmable logic array, PLA, and

associated analog electronics. This PLA first causes an ultrasonic signal to be emitted by the transmitters for both eyes and then it waits for a signal to be reflected back from an obstacle to either receiver. The time taken before the reflection comes back (assuming one does return) indicates the distance of the obstacle away from the nearest eye. The time taken is determined by the PLA.

An extra microprocessor circuit has been added to the insects which implements a Hopfield type neural network[2]. This circuit allows the insect to learn suitable behaviour, for example to move around an environment avoiding obstacles. This it achieves despite, or perhaps because of, its very simple sensory system. Therefore it was decided that a more advanced sensor was needed and the type of sensor chosen was determined by the fact that insects were being modelled.

2 The Compound Eye

Given the complex behaviours prevalent in the insect world, an investigation was made into their visual organs with the intention of building an electronic analogue. It was noted that many insects possess a pair of compound eyes, bulging out of the head with a wide field of vision. The eyes cannot move, have a short visual range and are of fixed focus. Each compound eye may contain upward of 10000 or more simple photoreceptive cells, each with its own lens. Most insect eyes respond to light between the wavelengths of 300mu to 600mu (orange), with near ultraviolet light being the most significant region for phototaxis (the insects visually related behaviours), with signals from the compound eye being directly related to such behaviours.

To investigate how useful a simple vision system might be to the Cybernetic insects, a small compound eye was built (see figure 1). A typical output image (normalised to the interval [0..255]) is shown in figure 2.

The eye consists of a three dimensional array of fifteen light dependant resistors (LDRs) mounted on the top fifteen faces of a truncated icosahedron. The input circuitry produces an output voltage dependent on the light falling on any one of the LDRs. An A/D converter gives a digital output of this voltage whilst an analogue multiplexer selects which LDR is currently being measured.

The half angle of the LDRs used is 120°, which gives good coverage but greater than optimal overlap between sensors. The discrimination of the eye could be improved by using sensors with a 60° half angle. The summed response of all sensors versus angle in the horizontal plane, along with typical responses of two adjacent sensors are shown in figure 3.

Figure 3a shows the response of the sensors used (120° half angle), figure 3b gives the response for a 60° half angle whilst figure 3c is for a 30° half angle. The arrows on figures 3a and 3b show the increased discrimination of the 60° sensor compared to the 120°. The summed response in figure 3c shows that reducing the receptive field of the sensors too much results in uneven coverage and hence increased sensitiv-

ity to small changes in light patterns within the environment.

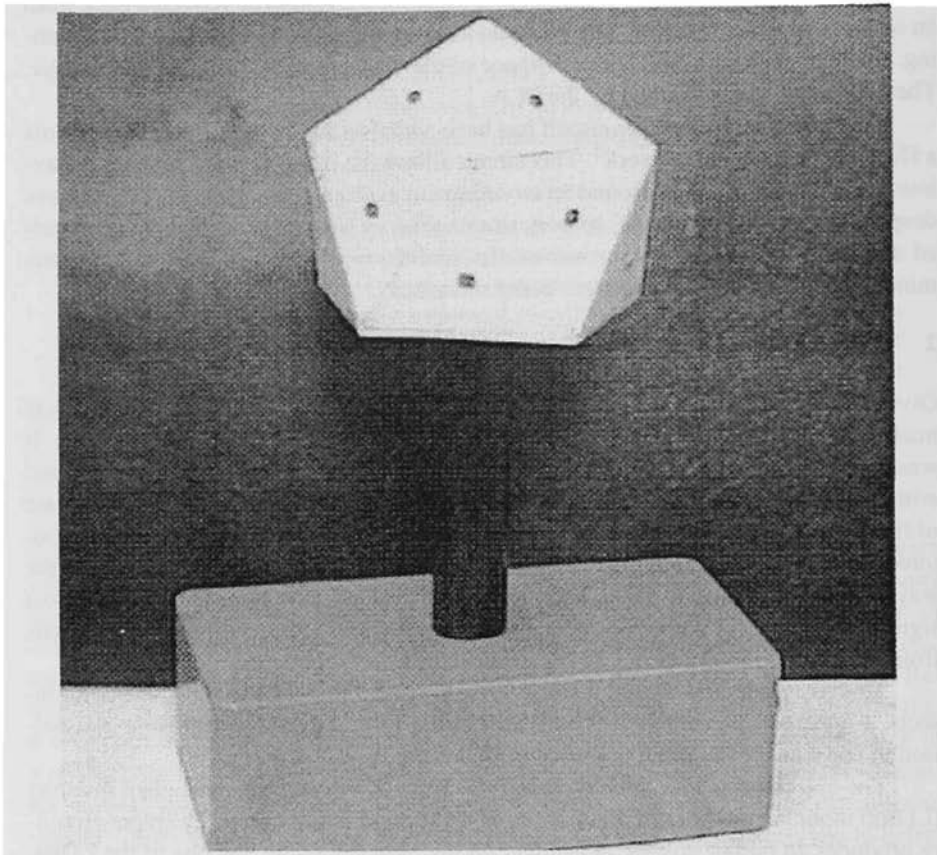


Figure 1. The compound eye

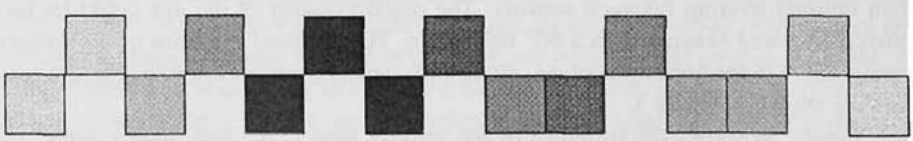


Figure 2. Typical 15 pixel image seen by eye

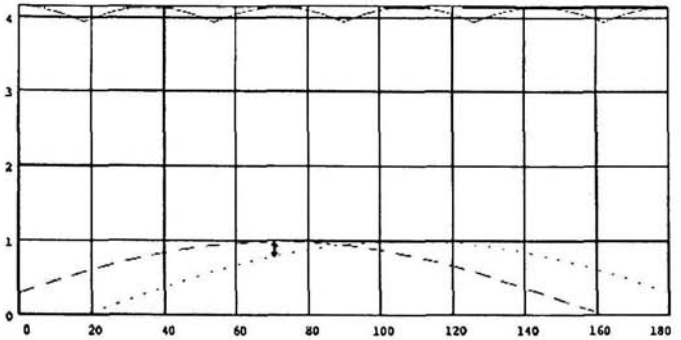


Figure 3a. 120 degree half angle

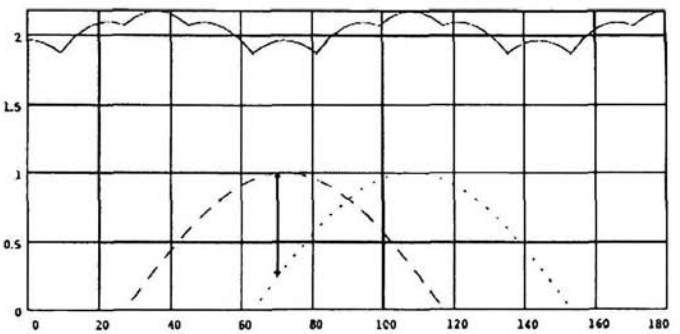


Figure 3b. 60 degree half angle

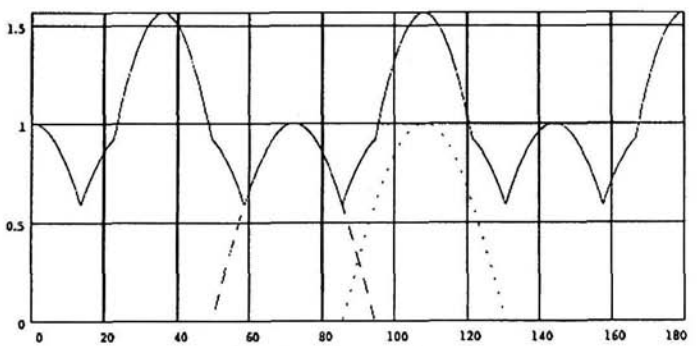


Figure 3c. 30 degree half angle

Figure 3. Half angle response of LDRs

3 The Use of a Weightless Network to Estimate Insect Position

The output from the compound eye consists of an array of fifteen values digitised to the interval [0..255]. Attempts to use a standard weightless network with thresholding proved unsuccessful, so the matrix was sampled by a large array of Minchinton Cells[3] whose output formed the retina of 1350 binary values presented to a 150 RAM multi-discriminator network.

To quantise an environment to one of n positions, n discriminators were used, with each discriminator being taught to respond strongly to the light patterns characteristic of a particular position within a 4×2m section of the lab. First experiments used sixteen discriminators in a [4×4] array, spaced at 0.5m intervals along the x-axis and 1m intervals along the y-axis. To achieve rotational invariance, each discriminator was taught at five rotations of 72° around this point (see figure 4).

A tuple size of nine was selected to give a sharp discriminator response. Each discriminator was trained with eleven sets of data collected at random intervals throughout a week. No attempt was made to normalise laboratory lighting, window blind positions or other activity in the lab. Hence the total number of patterns each discriminator was taught (after training rotations) was 55 (11×5), at which point average discriminator saturation was 7%.

This process was then repeated using the same data, but this time training four discriminators in a [2×2] regular array, spaced at 1m intervals along the x-axis and 2m intervals along the y-axis.

4 Results

The post training responses for an array of [4×4] and [2×2] discriminators are given in figures 5a and 5b. Each graph maps a discriminator response across the entire training area. Light areas correspond to regions of high response, dark areas, low response. Thus in figure 5a, where the experimental space is quantised over four regularly spaced discriminators (in the positions shown in figure 4a), the discriminator trained in the region at the top left of the area (Disc23) responds most strongly when in that location and its response decreases monotonically with distance from the training position. Similarly Disc22 responds best in the top right position, Disc21 in the bottom left and Disc20 in the bottom right. Comparable results (figure 5b) were observed for the sixteen discriminator system shown in figure 4b.

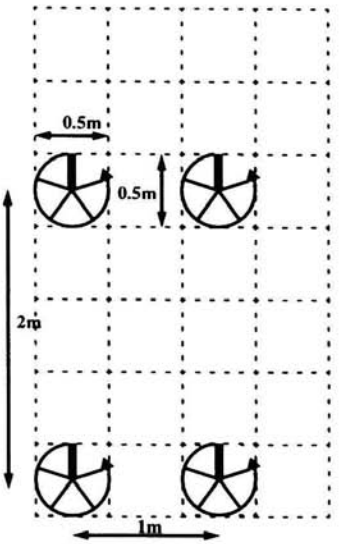


Figure 4a. 4 Discriminators

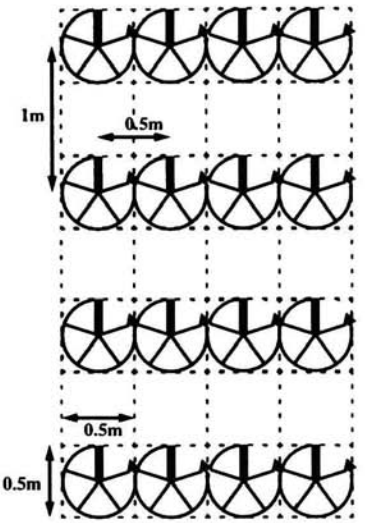


Figure 4b. 16 Discriminators

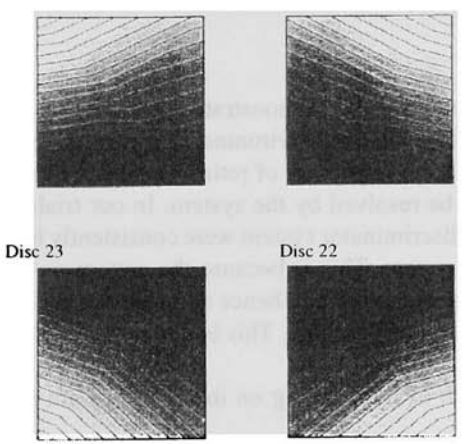


Figure 5a. Response of 4 discriminator network

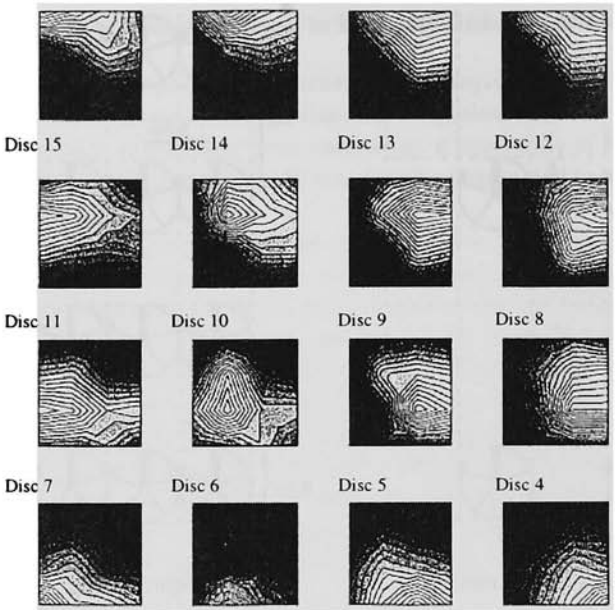


Figure 5b. Response of 16 discriminator network

5 Discussion

The results presented here have demonstrated that a simple compound eye can be used to resolve position in a complex environment. The initial experiments described herein have shown that for a given size of retina, there is a limit to the large scale spatial resolution that can be resolved by the system. In our trials, the Relative Confidence values for the four discriminator system were consistently higher than those of the sixteen discriminator system. This is because the pattern on the binary retina does not change much over a range of 0.5m, hence there is little useful information over which the discriminators can differentiate. This is due to the fact that:

a. The pattern of light falling on the eye does not change much over a small range.
b. An LDR half angle of 120° is not optimal for discrimination (see figure 3).

Changing network parameters, such as tuple size or degree of training, to make the network sensitive to such small changes reduces its ability to generalise over un-

seen data. Thus an important constraint in the design of an insect eye must be the spatial range over which it is required to operate.

The results obtained from the experiments reported here have led to the design of an improved eye with 32 light sensitive cells, each with a receptive field of 60°. This new eye will be used by the Cybernetic insects to guide themselves to the approximate location of an infra-red beacon (analogous to a food scent) from anywhere within the lab. Signals from the beacon then guide the insect to link up to a power supply and hence allow it to feed itself autonomously when it gets hungry!

References

1. R.J.Mitchell, D.A.Keating and C.Kambhampati, 1994, Proc. Control 94, pp: 492-497.
2. R.J.Mitchell, D.A.Keating and C.Kambhampati, 1994, Proc. EURISCON 94, pp: 78-85.
3. J.M.Bishop, P.R.Minchinton & R.J.Mitchell, 1991, Proc IMechE Conf, Euro-Tech Direct 91, pp: 187-199.

EXTRACTING DIRECTIONAL INFORMATION FOR THE RECOGNITION OF FINGERPRINTS BY pRAM NETWORKS

T. G. CLARKSON, Y. DING
Department of Electronic & Electrical Engineering, Kings College London, Strand, London WC2R 2LS, U.K.

A fingerprint processing and identification system based on pRAM neural networks is described. Firstly, a condensed numerical representation of a fingerprint is obtained which comprises a set of local directional images in two dimensions. Secondly, the core and delta points are identified, which are used as points of registration for the matching process. Finally, the input fingerprint can be matched with a set of reference fingerprints and the system displays the 10 best matching samples. The fingerprint is converted into a matrix of 17 × 17 directional images which are quantised to 8 levels. A two layer 6-pRAM pyramidal neural network was designed and trained by a reinforcement self-organising algorithm with an adaptive learning rate and Gaussian noise injection. The recognition rate is 86.4% for the best match. However, the system displays the 10 best matching samples so that these samples can then be manually inspected which is optimum for practical applications.

1 Conventional Methodology for Fingerprint Detection

The directional image is calculated from the original grey level fingerprint image and represents the local orientation of the ridge or valley of the fingerprint. To extract the directional image, the original grey level image **G** (size $M \times M$) is divided into several small blocks (of size $m \times m$). Obviously the number of blocks is $M/m \times M/m$. In our experiment M is 512 and m is selected as 16 because the average ridge thickness is around eight pixels in a standard NIST finger print image captured at a resolution of 512 dpi. A block size of 16 pixels will contain at least one ridge or one valley and hence has a characteristic direction. The orientation $D(i,j)$ of each block centred at (i,j) is the direction for which, the sum of difference in grey values along a direction n, is a minimum. That is:

$$D(i,j) = \min_{n} \sum_{k=1}^{L} \left| G_n(i_k, j_k) - \overline{m_n} \right| ; \quad n = 0, 1...N \tag{1}$$

where $G_n(i_k, j_k)$; $k = 1, 2,... L$ indicates the grey values of continuing points in one of the N+1 directions ($N = 7$) as shown in Figure 1. L is the size of the area (in pixels)

over which the directional image is measured and here $L = 17$. Note that L is an odd number to keep the neighbouring points symmetric about the central point (i, j). $\overline{m_n}$ is the mean grey value of pixels along each direction. Therefore from the grey level image $\mathbf{G}(512 \times 512)$, a directional image $\mathbf{D}(32 \times 32)$ is obtained by finding the predominant direction in each block of pixels.

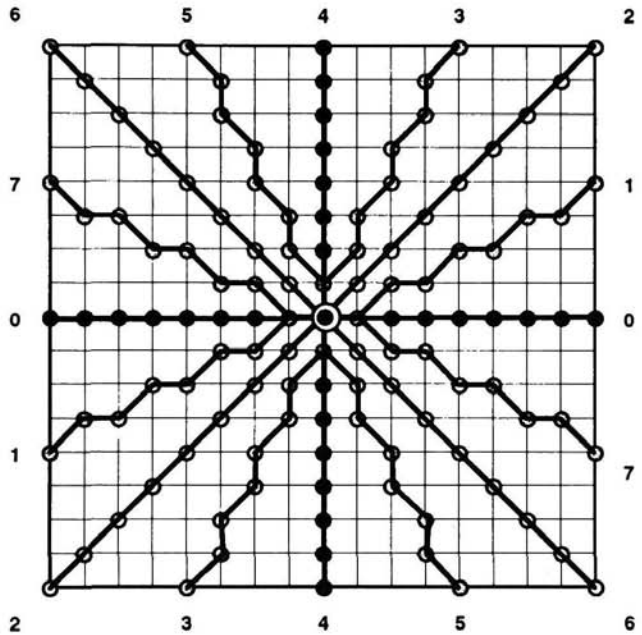


Figure 1. The sampling points for the orientation-feature inputs.

2 Algorithm based on pRAM neural networks

The algorithm above leads to some erroneous local vectors which will impair the classification accuracy of the system. This is due to the 17×17 sampling window having a limited field of view and results in aliasing effects. Fingerprint data contains many breaks in the ridges which can be interpreted as an orthogonal line passing through the ridge. For example, if a ridge shown in black has a break in it, certain alignments of the sampling window would show the predominant direction to be a white line orthogonal to the ridge and vice versa. Therefore, a neural algorithm was developed and trained on fingerprint data to improve the accuracy of the directional image.

In this algorithm we train a pRAM neural net as an orientation identifier. The

grey image is first converted into a binary image. A dynamic thresholding algorithm is designed to extract the binary image of the fingerprint, using the median value of a mask of size 17×17 as the threshold. Thresholding is performed by moving the mask over the image, and at each new position the new threshold value is determined for the pixel at the centre of the mask. In this way the blurring problem caused by a non-uniform background or different contrasts is avoided.

A two-layer pyramidal neural net was designed and trained as a classifier to implement a local valley and ridge orientation identifier *for each of the eight directions* in Figure 1. Note that ridges and valleys have the same significance in determining the directional image. The top two patterns in Figure 2 represent features which fall exactly along the axis sampled by the inputs of an identifier net. As shown in Figure 3, the first layer of one identifier net contains two 8-pRAM neurons; the output layer is a 2-pRAM neuron whose firing frequency is in the range of 0 to 1. Some standard patterns which are vectors in the $\{0, 1\}^{16}$ hyperspace, are used to train the pRAM net to output the corresponding code as indicated in Figure 2.

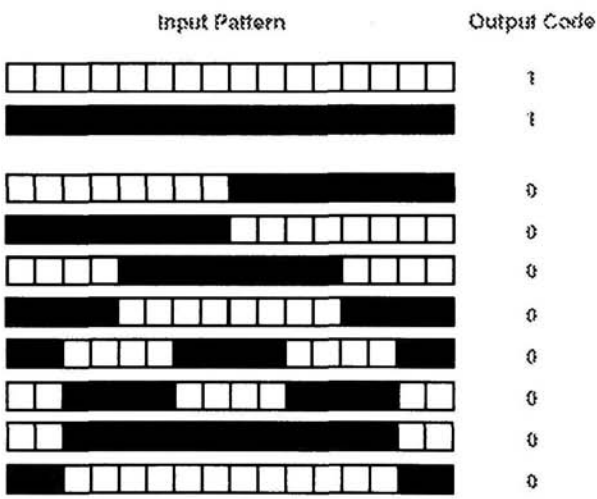


Figure 2. Training input patterns for the local orientation networks.

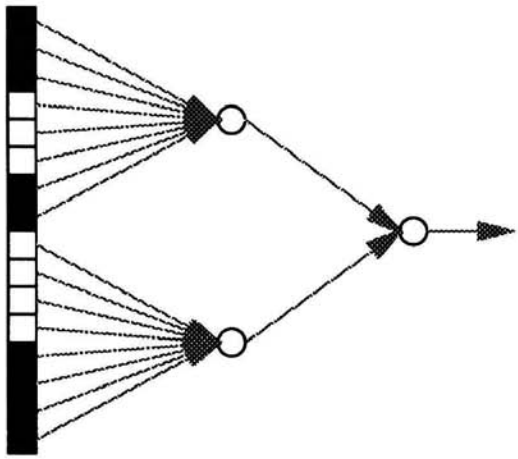


Figure 3. One of eight local orientation networks.

3 Training Algorithm

There are several training algorithms possible for pRAM neural nets, but we normally employ reinforcement training with noise enhancement[1,2]. In reinforcement training, an individual node only receives information about the quality of the performance of the network as a whole, and nodes have to discover for themselves how to change their behaviour so as to improve their performance[3]. The pRAM version of reinforcement algorithm is achieved by the update rule:

$$\Delta\alpha_{\underline{u}}(t) = \rho[(a-\alpha_{\underline{u}})r + \lambda(1-a-\alpha_{\underline{u}})p(t)] \times \delta_{\underline{u},\underline{i}} \qquad (2)$$

where

- ρ - reward rate $\in [0, 1]$ (typically 0.1),
- λ - reward to penalty ratio $\in [0, 1]$ (typically 0.05),
- a - output of the pRAM $\in \{0, 1\}$,
- $\alpha_{\underline{u}}$ - content of the memory location addressed by $\underline{u}$,
- E - error in the network' s output signal
- $r(E)$ - reward signal $\in [0, 1]$,
- $p(E)$ - penalty signal $\in [0, 1]$.

The Kronecker delta $\delta_{u,i}$ ensures that the update only occurs at the location which was accessed at time t. The above equation produces a new connection weight which is still in the normalised range 0 and 1, i.e., $\alpha_u^{new} \in [0, 1]$.

Given a training pattern I, if the output vector falls into the hypersphere with Hamming distance D from its centre v_c, where v_c has the maximal addressing probability from pattern I, then the reward signal is given to the network as a whole; otherwise the penalty signal is given.

It has been demonstrated that the generalisation ability of pRAM nets is greatly increased by injecting random noise into the training patterns[1,2] and optimum basins of attraction are formed. As the noise injected, the memory contents of all nearest neighbour vectors of any given pattern are modified so that they are more likely to fire or not to fire for that pattern or a similar one, which is how the generalisation behaviour is achieved. A training program in which the noise level starts from zero and is increased in steps of 5% was used. During training, the noise is increased until the success rate for any pattern falls below 80% so that maximum generalisation is obtained.

3.1 Decision Strategy

The firing rate of the neuron in the output layer is employed to judge the local fingerprint valley or ridge orientation in a 17×17 window. As shown in Figure 1, the binary value of pixels in each of the eight directions, except the central pixel value, is a 16-dimensional vector which is input into the net. The firing rate of the vector in each direction is denoted as Fr(i), $i = 0, 1, ..., 7$; where i indicates the direction as in Figure 1. Suppose the two largest values of Fr(i) are Fr(t) and Fr(r) respectively, Fr(t) > Fr(r). *dir* denotes the orientation in the block; if Fr(t) - Fr(r) > 0.15 and Fr(t) > 0.70. That means the firing rate in direction t is much greater than that in any other direction and there is sufficient confidence to judge t as the direction of the local valley or ridge. Otherwise the directional vectors in the eight neighbouring windows are estimated and summed for each direction and *dir* is assigned the orientation given by the maximum of the eight summed responses.

4 Testing the direction-extracting network

The local ridge or valley orientations in a right thumb fingerprint, f0001_01.pct in the NIST fingerprint database, were determined by a conventional algorithm and a pRAM neural net respectively[4]. The pRAM orientation detector estimated the local ridge or valley direction more accurately than the conventional method. Although the pRAM method is more time-consuming using software simulation, this can be compensated by the hardware implementation of a pRAM net[5]. The directional image of the fingerprint is used for patch matching and core and delta point detection as shown in a later

section.

5 Region of Interest (RoI) Detection

Any fingerprint recognition system must start with identifying a pair of fingerprint images. The system outputs an estimated value of the probability that the two images originate from the same finger. However we must first decide the matching RoI of a pair of fingerprint images, in which the effective features indicating the difference between the two fingerprints are extracted and input to the pRAM classifier. For a pair of grey level images (we call the first fingerprint image the *target* image and the second is called the \it *test* image), the directional images of size 32×32 are calculated by the above method. Then the 15×15 central region of the target directional image, denoted D_1, is used to search the same size optimum matching region in the examination directional image, denoted D_2, within a larger 24×24 patch by estimating the matching value ζ pixel by pixel. ζ can be estimated by the following three methods:

5.1 Correlation Coefficient

We define ζ to be the correlation coefficient between the two directional images and decide the area giving the best match by finding the maximum value of ζ.

$$\zeta = \frac{\sum_{j=0}^{14}\sum_{i=0}^{14} D_1(8+i, 8+j) \times D_2(3+I_0+i, 3+J_0+j)}{\sqrt{\sum_{j=0}^{14}\sum_{i=0}^{14} D_1(8+i, 8+j)^2}\sqrt{\sum_{j=0}^{14}\sum_{i=0}^{14} D_2(3+I_0+i, 3+J_0+j)^2}} \tag{3}$$

where D_1 (i,j) and $D_2(i,j)$ are the orientation values of pixels in D_1 and D_2 at address (i,j) respectively, and ζ is between 0 and 1. The bigger the value of ζ, the more similar the RoI given by the top left vertex $(3+ I_0, 3+J^0)$ in D_2 to the central RoI decided by the top left vertex (8, 8) in D_1 and vice versa.

5.2 Mode Difference

We define ζ as the sum of the orientation difference between the collateral pixels in the RoI of the two directional images and decide the area of best match with respect to the top left vertex at $(3+ I_0, 3+ J_0)$ in D_2 by finding the minimum value of ζ.

$$\zeta = \sum_{j=0}^{14} \sum_{i=0}^{14} [D_1(8+i, 8+j) \Theta D_2(3+I_o+i, 3+J_o+j)]^2 ; \quad (4)$$

$$I_0 = 0, 1, \ldots, 10; J_0 = 0, 1, \ldots, 10; \quad (5)$$

where $a \Theta b$ means:
if (abs(a - b)) $\geq$ 5,
$a \Theta b = a - b$
else {
if ($a > 4$) $a = a$ mod 4;
if ($b > 4$) $b = b$ mod 4;
$a \Theta b = a - b$;
}

5.3 *Directional Histogram*

The 15×15 pixel RoIs in both D_1 (with the top left vertex at (8, 8)) and D_2 (with the top left vertex at (3+ I_0, 3+ J_0) are divided into nine equal size squares in which the directional histogram $H_j^k(i) = 0, \ldots 7; j = 0, 1, \ldots 8; k = 1, 2$ is generated, where i denotes the eight directions, j indicates the nine areas and k indicates the histogram in D_1 or D_2.

We define

$$\zeta = \sum_{j=1}^{8} \sum_{i=1}^{7} [H_j^1(i) - H_j^2(i)]^2 \quad . \quad (6)$$

the best matching areas are decided by finding the minimum value of ζ.

Our experiments have shown that the Directional Histogram method is the optimum approach both by means of visual inspection of the matching RoI and the final recognition result. One example is shown in Figure 4, where the central directional RoI in D_1 (the top directional image) is used to find the most matching region in D_2 (the bottom directional image). Sometimes the matching value of nonmatching pairs is lower than that of matching pairs (by the Directional Histogram method) Therefore non-linear recognition by a neural network algorithm is necessary.

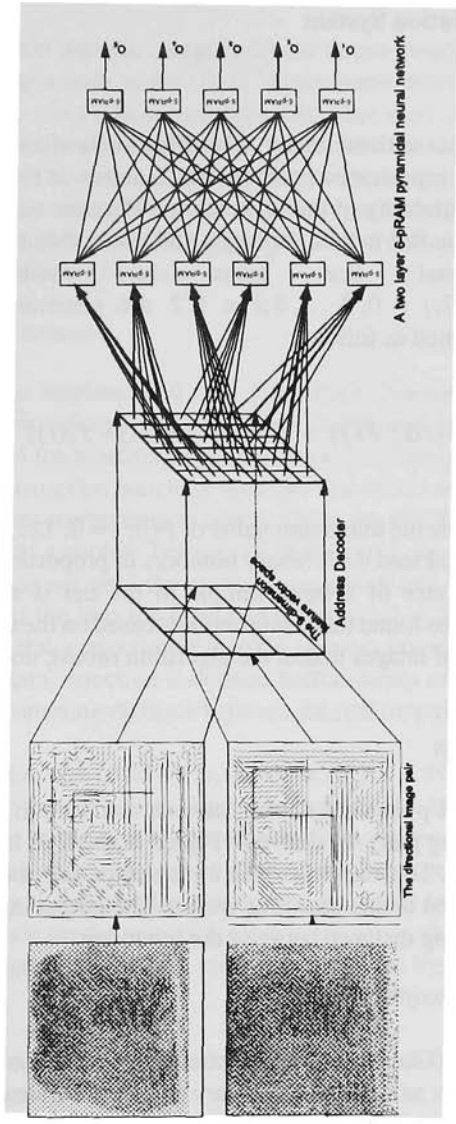


Figure 4. The fingerprint classification network.

6 pRAM Classification System

6.1 Input data

Feature selection is not a trivial matter. Classification is often more accurate when the pattern is simplified represented by important features or properties[3]. In our feature detector, only the variability of the orientation difference statistics is considered. As shown in Figure 4, the RoI in both the target and tested directional images are divided into nine equal size areas in which the histograms $H_j^k(i); i = 0, 1, \ldots, 7; j = 0, 2, \ldots, 8; k = 1, 2$ are generated. So the 9-dimension feature vector is defined as follows:

$$f(j) = F(j)/\alpha;\ F(j) = \zeta = \sum_{i=1}^{7} [H_j^1(i) - J_j^2(i)]^2;\ j = 0,1,\ldots,8; \qquad (7)$$

where $\alpha = \max / 15$ is the maximum value of $F(j)$; $j = 0, 1,\ldots, 8$; Then the input features $f(j)$ are converted into 4-bit binary numbers in proportion to their value. Therefore the total input size of a input sample to the net is a vector in the $\{0,1\}^{36}$ hyperspace. It has been found that using features based on the difference of histograms in a pair of directional images makes the algorithm robust, noise tolerant and computationally efficient.

6.2 Classifier design

A two-layer 6-pRAM pyramidal neural network was used to classify fingerprints by identifying a matching pair. As shown in Figure 4, the first layer contains 6 neurons and each has 6 inputs. The mapping of the input vector onto the input layer can be random, the mapping used in this case is as shown. The output layer uses five neurons to represent the Hamming distance between the input pair.

6.3 Training the classifier

We employed a GNI (Gaussian noise injection) generalisation method[6] which is used for real-valued inputs as opposed to binary inputs. Euclidean distance, rather than Hamming distance, is employed to indicate the similarity of the input patterns. The training algorithm is uses a direct binary coding of the real-valued input with reinforcement training and Gaussian noise injection. When an input vector is presented to the net in both training and testing phases, not only is the addressed memory content accessed, but also a radial region in Euclidean distance from that input. The Hamming distance between the output code and the target output code is used to develop the re-

ward and penalty inputs.

A sample is classified into the category whose target output code is the nearest neighbour to the real output code in the $\{0, 1\}^n$ hyperspace, where n is the number of output nodes (in this case $n = 5$). In our experiments, there were many less samples of the matching pairs than of nonmatching pairs. To prevent the network training from becoming biased towards those classes which contain larger number of training samples, an adaptive learning rate in which the *a-priori* class distribution is considered was adopted in our algorithm where a learning rate ρ in inverse proportion to the number of patterns in each class was employed[7].

7 Test Results and Discussion

The above approach was implemented on a Sun SPARCStation in a Khoros image processing software environment. The directional image was extracted by a pRAM neural net algorithm and the matching patches of the two fingerprint were found by the above Directional Histogram matching method. The neural network classifier was trained by 322 pairs of fingerprint images selected from the NIST fingerprint database, and tested by 220 pairs of samples. Training of the network was continued over several epochs of the training set until the correct recognition rate was over 95%, when the standard deviation of the injected Gaussian noise is 2.0. A Reject class was used in the system, as in general a rejection is thought of more favourably than a misclassification. In our algorithm, rejection is defined both in terms of the firing rate of the output neuron and the Hamming distance between the real output and the target output code of each category[6].

A correct recognition rate of 86.4%, a rejection rate of 4.5% and a failure rate of 9.1% were obtained for the testing set. For classification of a biased sample set, this result is quite promising. In addition to classification, the system can output the 10 best matching samples to the target fingerprint by recording the mean firing frequencies of each output node so that these samples can then be manually inspected. Suppose the desired binary output code for the matching set is $o_1, o_2, \ldots, o_n$, the probability P of one examination sample matching the target sample is estimated by:

$$P = \prod_{i=1}^{n} (p(o_i)o_i + p(o_i)o_i) \tag{8}$$

where $p(o_i)$ is the firing rate of the ith neuron in the output layer of the neural network. 48 target images were used in this test to search the matching pairs in a database which includes 400 samples. The order of the matching values (from the largest to the smallest) of the desired pairs are reported in Fig. 5. We found that the matching values of correctly-classified pairs were always listed in the ten best matches and in 68.7% of

cases, the P value of a correct pair was the highest output. This result has value in practical applications as it greatly decreases the amount of manual inspection required while maintaining a high accuracy rate. This demonstrates that the neuron firing rate is a effective way to estimate the similarity of the patterns. In the practical system only the central (24×24) directional images need to be stored in the database so the costs in terms of both memory space and time are greatly reduced. To reduce the error rate in identifying matching pairs, one possible way is to input another 9-dimension vector which is extracted by a similar approach but uses a pair of RoIs which possess mirror symmetry to the original pair of ROIs. Suppose the original (15×15) RoI pair is referenced to the top left vertex (8, 8) in D_1 and (I, J) in D_2, so the mirror symmetry RoI can be determined by the top left vertex $(8 + (8 - I), 8 + (8 - J))$ in D_1 and (8, 8) in D_2. In this way, the binary input will be in $\{0,1\}^{64}$ hyperspace. However, the resulting neural nets will be more complicated. Research continues into increasing the number of features or selecting new features and making the system robust by testing distorted fingerprint images in a larger database.

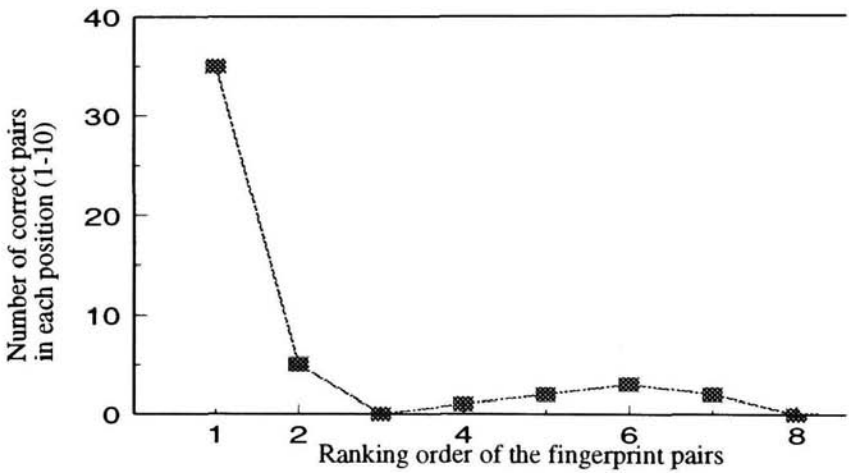


Figure 5. Distribution of correctly-matched pairs in the top ten scores.

References

1. Y Guan, T G Clarkson, D Gorse and J G Taylor, Noisy reinforcement training for pRAM nets, Neural Networks, **7**, 1994, 523-538.
2. T G Clarkson, Y Guan, D Gorse and J G Taylor, Generalisation in probabilistic RAM nets, IEEE Transactions on Neural Networks, **4**, 2, 1993, 360-364.
3. Y Ding, Y Guan, T G Clarkson and R P Clark, The Back Thermal Symmetry

Identification by pRAM Neural Networks, Proc. ANN'95, IEE 4th International Conference on Artificial Neural Networks, Cambridge, 1995, 437-441.

4. Y Ding Y and T G Clarkson T G, Extracting the directional image of fingerprints by a neural algorithm, WNNW'95 Canterbury, Sept 1995, 52-56.
5. T G Clarkson, D Gorse, J G Taylor, C K Ng, Learning Probabilistic RAM Nets Using VLSI Structures, IEEE Trans. Computers, **41**, 12, 1992, 1552-1561.
6. Y Ding and T G Clarkson, A novel scheme to develop generalisation in a pRAM neuron with a real-valued pattern feature input, Submitted to IEEE Trans. on Neural Networks.
7. S Ramanan, R S Petersen, T G Clarkson and J G Taylor, Adaptive learning rate for training pyramidal pRAM nets, Proceedings of ICANN'94, Marinaro and Morasso (Eds),1994, 1360-1363.

DETECTION OF SPATIAL AND TEMPORAL RELATIONS IN A TWO-DIMENSIONAL SCENE USING A PHASED WEIGHTLESS NEURAL STATE MACHINE

P. NTOURNTOUFIS[†], T. J. STONHAM
Dept. Electrical Eng. & Electronics, Brunel University, Uxbridge, Middlesex UB8 3PH, UK

The detection of spatial and temporal relations in a scene is best illustrated by considering a mobile robot which can change the visual image it receives by moving the position of its visual sensor. The problem is then to show how such a robot can utilise its actions, moving the sensor, to get proper indexing of the visual information and so encode spatial relationships linguistically. The preliminary problem addressed in this chapter is a simpler one. The robot's visual sensor is directed to specific locations in a two-dimensional scene where various objects are located. For each object, a linguistic label is provided, describing the class of the object in question. The robot's task is to learn to search in the scene for objects by name.A novel two-phase configuration of a MAGNUS (Multi-Automata of General Neural UnitS) weightless neural system is used to carry out the investigation. A training procedure which enables the network to perform the given task optimally is presented.

1 A Two-phase MAGNUS Network

The MAGNUS is a general weightless neural system used in areas such as language and two-dimensional representations of three-dimensional scenes. It is a learning and generalising device in which the training set forms attractors or attractor trajectories in state space. It is able, for example, to relate language-like symbol strings to internal iconic representations and respond to environmental input with appropriate actions.[1] This capability of internal representation of informational structures distinguishes MAGNUS from earlier learning devices[2] which merely performed pattern recognition.

The neural processing elements of the system are Generalising Random Access Memories (GRAMs).[3] A G-RAM accepts binary inputs and produces a binary output. During training, input-output associations are recorded. Generalisation in the G-RAM is provided by some spreading procedure.[4] During recall, the G-RAM uses the stored

† Current address: *Anite Systems Ltd, Finance Division, Gavrelle House, 2-14, Bunhill Row, London EC1 8LP, UK*

information to produce outputs, not only for a trained input pattern but also for other input patterns that are closer to it, in Hamming distance, than to any other training pattern. Input patterns which are equidistant from training patterns and which require different outputs, will produce an output 1 or 0 with equal probability.

The core structure of the MAGNUS is that of a neural state machine (NSM)[5] , with well defined sensory inputs.[6] In this chapter, two types of sensory inputs are used, visual and "linguistic". The training of the network is iconic,[5,6] enabling the creation of internal states that are directly linked to sensory experience. The NSM structure provides the system with the ability to create an internal state structure which is representative of the system's environmental experience.

MAGNUS interacts with TableTop, a near-real-world model consisting of a two-dimensional projection of a three-dimensional scene, containing cups, glasses, mugs and plates. The resulting image (which, in the present case, has dimensions 512×512) has a wrap-around topology, eliminating the problem of borders. Visual sensory input to MAGNUS is provided by a central retinal window (RW), acting on the visual scene. Functions performing panning (x), scrolling (y) and zooming (z)[7] of the RW are implemented in the TableTop model. Fig. 1 shows the structures of the MAGNUS system and the TableTop near-real-world model.

The MAGNUS system is characterised by a multi-field neuronal structure. Five such fields can be distinguished, corresponding to the parts of the network coding for the visual (F_V) and linguistic (F_L) internal representations, and for the 3 motor outputs of the network (F_X, F_Y and F_Z). The latter determine the position and size of the RW at the next moment in time. A thermometer coding scheme is used to transform the outputs of F_X, F_Y and F_Z into integer RW co-ordinates x, y and z.

Two types of feed-back exist in the network: an *internal* feed-back, between the neural field outputs and the network inputs, and a *external* feed-back, between the motor outputs of the network and the RW. This second level of feed-back characterises structures which implement an attention mechanism[8].

A two-phase configuration of the network is chosen. The fields F_V and F_L receive as input both the visual and linguistic inputs, as well as their output states S_V and S_L at the previous moment in time. The output fields F_X , F_Y and F_Z receive as input the output states S_V and S_L , as well as their output states X , Y and Z at the previous moment in time.

2 Finding Objects by Name

A two-dimensional scene contains a number of objects (cups, glasses, ...) belonging to one of Z classes $C_1, C_2, ..., C_Z$. The number of objects belonging to class C_c is denoted J_c. In the scene represented in Fig. 1, $Z = 4$, $J_{cup} = 2$, $J_{glass} = 3$, $J_{mug} = 2$ and $J_{plate} = 5$. A linguistic input L_c , representing the name of an object of class C_c is presented to the network, which is required to move the RW to a position in the scene correspond-

ing to that of an arbitrary object of class C_c. The visual input to the network corresponding to object j of class C_c is denoted I_{jc}. The patterns coding the co-ordinates (x, y, z) of object j of class C_c are denoted X_{jc}, Y_{jc} and Z_{jc}, respectively.

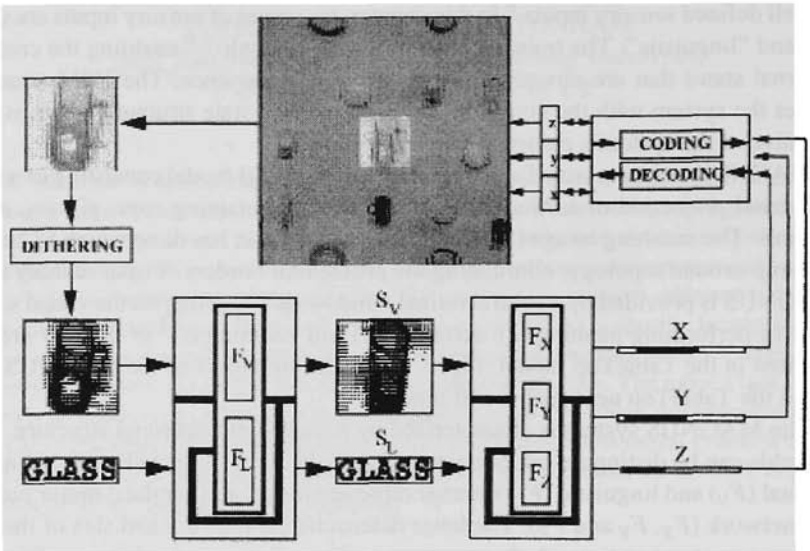


Figure 1: The structures of the MAGNUS system and the TableTop near-real-world model.

3 Network Training

All network fields are trained by storing input-output associations in their G-RAMs. In the case of a field with partial feed-back from its own output, any training association can be conveniently expressed as a triplet $\{E,P,S\}$, where E represents the external input, P the previous state and S the current state. In the case of a stable state creation, P and S are identical and such a training association is denoted $\{E,(S)\}$. Furthermore, if the training is iconic, and are identical and the resulting association is denoted [S].

The training is divided into 3 stages. During the first stage, each object of each class, in association with the corresponding linguistic input, is trained *iconically* so as to form a stable state representation at the output of fields F_V and F_L:

$$F_V F_L\colon\ \forall j, \forall c\colon \{[(I_{jc}, L_c)]\} \tag{1}$$

In the second stage, each object name is trained *iconically* so as to form a stable state at the output of F_L, when the visual input represents an object belonging to a different class. This stage enables the network to switch its internal representations between object classes:

$$F_L\colon \forall j,c,d \mid c \neq d : \{(I_{jc}, [L_d])\} \tag{2}$$

During the third stage, the previously trained internal representations are associated with thermometer codes of co-ordinates of objects belonging to the same class, through the training of the fields F_X, F_Y and F_Z:

$$F_X F_Y F_Z\colon \forall c,j,k : \{I_{jc} L_c, (X_{kc}, Y_{kc}, Z_{kc})\} \tag{3}$$

Eqs. 1 to 3 show that the number of input-output associations necessary to train the network is very small. For the first training stage, both F_V and F_L each require J training associations. For the second training stage, F_L requires $J \cdot (C-1)$ training associations. Finally, during the third training stage, F_X, F_Y and F_Z each require $\sum_{c=1}^{C} J_c^2$ training associations. In the case of the scene shown in Fig. 1, fields F_V and F_L and require 12 and 48 training patterns, respectively; and fields F_X, F_Y and F_Z require each 42 training patterns.

4 Recall

During a network run or recall, a linguistic input L_c is presented to the network and remains unchanged during the duration of the run. Instead of recalling the outputs of all fields synchronously, fields F_V and F_L are recalled a number of times first. This is followed by a number of recalls of fields F_X, F_Y and F_Z. Before the first recall of fields F_X, F_Y and F_Z, noise is added to their feed-back inputs by setting the X, Y and Z states to thermometer codes representing random co-ordinates. Finally, after the recalls of fields F_X, F_Y and F_Z, a move of the RW is performed.

The asynchronous updates of the network fields enables a greater flexibility in the specification of the network dynamics than in the case of a fully synchronous system, yielding an improved state structure. In the present experiment, fields F_V and F_L are updated five times. This is followed by five updates of fields F_X, F_Y and F_Z. This scheme allows the visual and linguistic internal states to settle into stable states before they are seen by fields F_X, F_Y and F_Z. Similarly, the states X, Y and Z are allowed to settle into stable ones before an actual retinal move is operated.

Fig. 2 shows characteristic inputs and internal states during a run of the network. The network contains 5872 64-input G-RAMs: fields F_V and F_L contain each 2304 G-RAMs; fields F_X, F_Y and F_Z contain 512, 512 and 240 G-RAMs, respectively. The initial internal states S_V and S_L are chosen to be random. The word "CUP" is presented at the linguistic input terminal. After several recalls of fields F_V and F_L, the states and have settled to the visual and linguistic internal representations of a cup. This is followed by several recalls of fields F_X, F_Y and F_Z which enable the states X, Y and Z to settle to thermometer codes representing the location of a randomly chosen cup in TableTop. This location is not necessarily that of the cup which produced the current state S_V. This is a consequence of training Eq. (3) and of the initial random thermometer codes X, Y and Z.

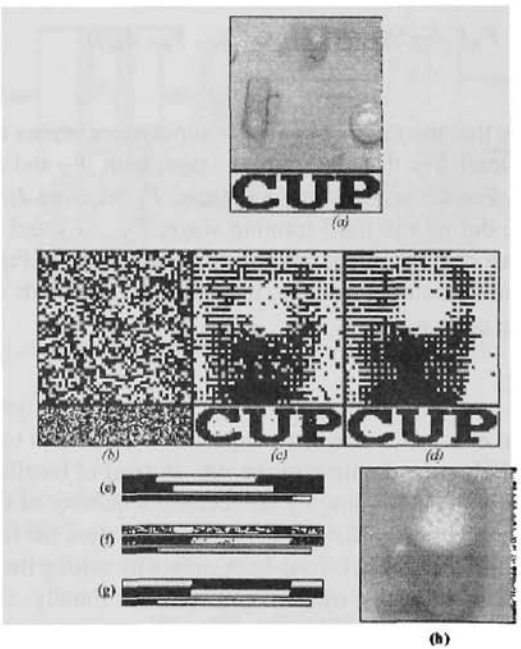


Figure 2: (a) Initial visual and linguistic inputs. States S_V and S_L, (b) before any recall; and after (c) one and (d) five recalls of the fields F_V and F_L. States X, Y and Z after (e) a random move, (f) one and (g) five recalls of fields F_X, F_Y and F_Z. (h) Final visual input.

Fig. 3 shows characteristic inputs and internal states during a next run of the network. The word "CUP" is replaced by the word "PLATE" at the linguistic input ter-

minal. After several recalls of fields F_V and F_L, the states S_V and S_L have settled to the visual and linguistic internal representations of a plate. This is followed by several recalls of fields F_X, F_Y and F_Z which enable the states X, Y and Z to settle to thermometer codes representing the location of a randomly chosen plate in TableTop.

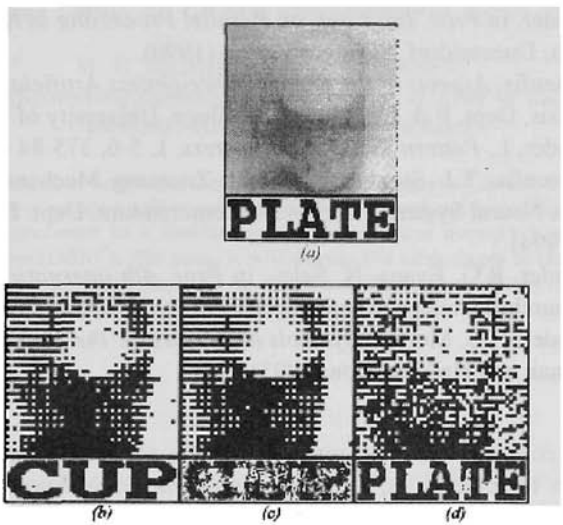


Figure 3: (a) Initial retinal and linguistic inputs. States S_V and S_L, (b) before any recall; and after (c) one and (d) two recalls of the fields F_V and F_L.

5 Conclusions

The problem of training a robot to search objects by name in a two-dimensional scene has been addressed and successfully solved. This is a preliminary step to the more general problem of detecting spatial and temporal relations in the scene. A novel two-phase configuration of the MAGNUS weightless neural system and a training procedure have been demonstrated which yield a trained network performing the task optimally.

Acknowledgements

This research was supported by the UK Engineering and Physical Sciences Research Council under grant no. GR/J15032. The authors would like to thank the other members of the MAGNUS project: Igor Aleksander, Thomas Clarke, Richard Evans, Nick Sales and Manissa Wilson.

References

1. I. Aleksander, R. Evans, and W. Penny, in *Proc. Weightless Neural Network Workshop'93*, pp. 156-9, York (1993).
2. I. Aleksander, W.V. Thomas, and P.A. Bowden, in *Sensor Review*, **4**, 3, pp. 120-4 (1984).
3. I. Aleksander, in *Proc. Int. Conf. on Parallel Processing in Neural Systems and Computers*, Duesseldorf, Springer Verlag (1990).
4. P. Ntourntoufis, *Aspects of the Theory of Weightless Artificial Neural Networks*, Ph.D. Thesis, Dept. E & EE, Imperial College, University of London (1994).
5. I. Aleksander, I., *Pattern Recognition Letters*, **1**, 5-6, 375-84 (1983).
6. P. Ntourntoufis, T.J. Stonham, "A Fast Zooming Mechanism for a Magnus Weightless Neural System", Technical Memorandum, Dept. EE&E, Brunel University, (1994).
7. I. Aleksander, R.G. Evans, N. Sales, in *Proc. 4th international conference on ANN's*, Churchill College, Cambridge (1995).
8. I. Aleksander, H.B. Morton, *Symbols and Neurons: the Stuff That Mind is made of*, Chapman and Hall, London (1993).

COMBINING TWO BOOLEAN NEURAL NETWORKS FOR IMAGE CLASSIFICATION

A. C. P. L. F. DE CARVALHO
Computing Department, University of São Paulo at São Carlos, São Carlos, SP CP 668, CEP 13560-970, Brazil

M. C. FAIRHURST, D. L. BISSET
Electronic Engineering Laboratories, University of Kent at Canterbury, Canterbur, Kent, CT2 7NT, England

This chapter describes and evaluates a completely integrated Boolean neural network architecture, where a self-organising Boolean neural network (SOFT) is used as a front-end processor to a feedforward Boolean neural network based on goal-seeking principles (GSN^f). For such, it will discuss the advantages of the integrated SOFT-GSN^f over GSN^f by showing its increased effectiveness in the classification of postcode numerals extracted from mail envelopes.

1 Introduction

The concept of modularity is a current issue in many research areas such a computer science and electronic engineering. The division of complex, larg tasks into simpler, smaller subtasks facilitates the understanding and the so lution of many difficult problems. The design of modular neural networks very important for future development of more complex neural networks. A cording to Houk [1], modular networks may be motivated on neurobiologic principals, once modularity seems to be an important feature in the archite ture of vertebrate nervous systems. A complex task, like image recognitio usually involves a number of smaller sub-tasks which have to be appropriate tackled in order to allow its full accomplishment.

The existence of hierarchical representations in the visual areas is sup ported by Fodor [2]. As stated by Treisman [3], a large number of psychologis and physiologists are agreed that the visual processing of information can b divided into two or more stages. In the initial stages occur the identification simple properties or features such as colour, size, orientation of lines, closur curvature and line ends. The next stages are then concerned with the comb nation of these features in order to describe and recognize scenes and object Integrated artificial neural architectures following this philosophy have bee successfully investigated by a large number of researchers [4,5,6,7,8].

In this chapter, the task of machine printed character recognition is divide into two smaller tasks, feature extraction and pattern classification. Each

these tasks is carried out by a Boolean neural network architecture. These networks, SOFT^d and GSN^f, are integrated in a complete Boolean architecture for image recognition.

The main idea behind the development of SOFT^d is to provide a front-end processor to GSN-based neural networks. According to[7], if an architecture is to be used as a front-end processor, it should present two main attributes. First, the task performed by the pre-processor should be both useful and significant. Second, the pre-processor output should be compatible with the input required by the higher level processors. The output provided by SOFT^d is completely compatible with the input required by the higher level processors, here the GSN^f architectures.

In this integrated architecture, the input image is first pre-processed by an already trained SOFT^d to be then directly used, without further pre-processing, as input by GSN^f. A SOFT^d architecture has been used instead of the original SOFT architecture because of its better pre-processing, as was shown in [9]. The main difference between SOFT and SOFT^d is the dynamic definition of the architecture used by SOFT^d, where nodes can be either created or eliminated during the learning process. Experiments are carried out in this chapter in order to evaluate the influence of the use of the pre-processed input data provided by SOFT^d architectures on the performance obtained by GSN^f networks. The following section describes the main features of GSN^f neural architectures.

2 GSN^f Neural Networks

GSN^f uses goal-seeking neurons (GSNs), which are similar to Random Access Memory devices. Each node (the term node will be used instead of neuron throughout this chapter) uses the input values presented to its input terminals to access one or more of its 2^N memory contents, where N is the number of input terminals.

GSN was designed to overcome many of the problems found in other Boolean models, maximising the efficiency of storage of values in its memory, storing new information without disrupting previously absorbed information, and employing one-shot (single-pass) learning. The GSN can accept, store and generate values of 0, 1 and u, where u represents an "undefined" value. If there is at least one undefined value on the input terminals of a GSN unit then a set of memory contents, rather than a single storage location, will be addressed, and this is a principal factor in the efficient distribution of stored information across a trained network[10].

GSN nodes have been emploied in three different architectures, each one

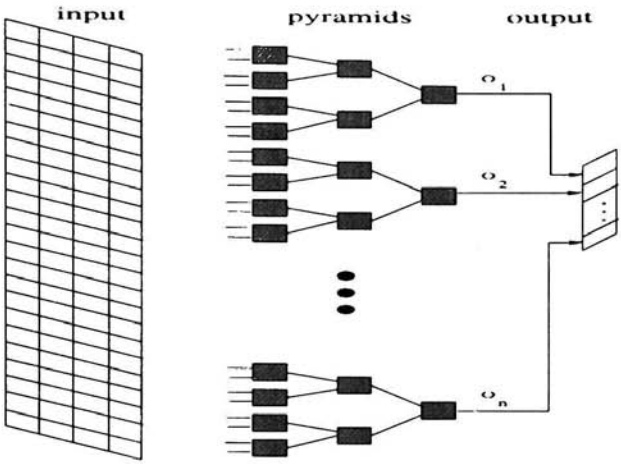


Figure 1: GSN^f architecture

adequate for a different scope of applications. These architectures are GSN feedforward (GSN^f), GSN feedback (GSN^{fb}), and GSN self-organising (GSN^s) [11]. Figure 1 shows the GSN-based architecture used throughout this chapter, which is the GSN^f model [12].

As can be seen Figure 1, GSN^f is a multi-layer feedforward architecture where GSN nodes are organised into several independent pyramids. Its processing is performed in three different phases. These are a validating phase, which defines which pyramids can be trained, a learning phase where, by using a supervised learning algorithm, defined values can be stored in memory locations holding undefined values, and a recall phase, where the node generates as output the predominant value stored in the memory contents addressed from the input. Next section describes another Boolean neural network, SOFT^d, which is used as a front end processor in the integrated architecture.

3 Feature Extraction by SOFT^d

Using a lattice structure, SOFT^d (Self-Organising Feature Extractor) relies on self-organising properties of local rules at a node level to change the memory content of the nodes and thus the overall function of the network. The design principles adopted are such that they guarantee a fast processing rate with minimum use of memory while ensuring low connectivity, high independence, easy parallel implementation and simple learning and recall rules within the

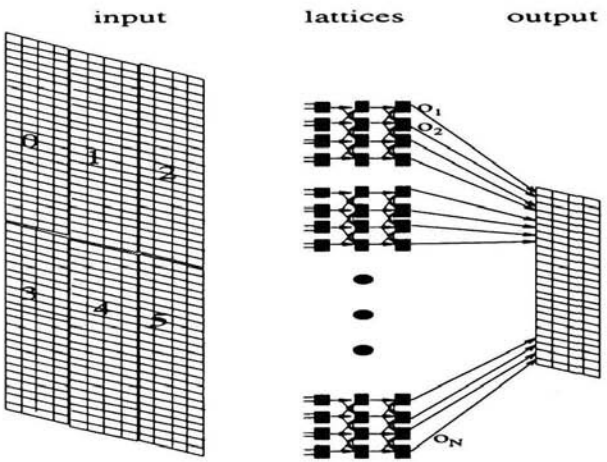


Figure 2: SOFT^d architecture

network. Figure 2 illustrates a typical SOFT^d network.

Using GSN-like nodes, SOFT^d is composed of a number of blocks, where each block covers a sub-area of the input image. As can be seen in Figure 2, its connection mapping provide more than one path from a given pixel in the input to a given output node. This ensures the migration of data across the network and in turn allows the extraction of features filling the input space without high node connectivity.

Figure 3 illustrates the division of the input image into six sub-images or areas, where each area covers an 8x8 group of pixels. In the experiments illustrated in this chapter, a three-layer node lattice is associated with each area, where each layer has initially 16 nodes, each with 5 input terminals and either 1 output terminal (when it is the output layer), or 5 output terminals otherwise.

SOFT^d uses an unsupervised learning algorithm where it teaches its blocks in parallel. The layers of each node block can be trained in a parallel pipeline, starting with the nodes in the first layer and, using the output produced by the already trained previous layer as input, the subsequent layers can be trained. Its learning rule defines the value to be stored in each addressed content $m_i \in M$ of each node by selecting, for each node, two memory attractors, which are memory contents with complementary addresses.

These attractors are selected by dividing the node input space into two halves in such a way that the node input patterns with a lower frequency in

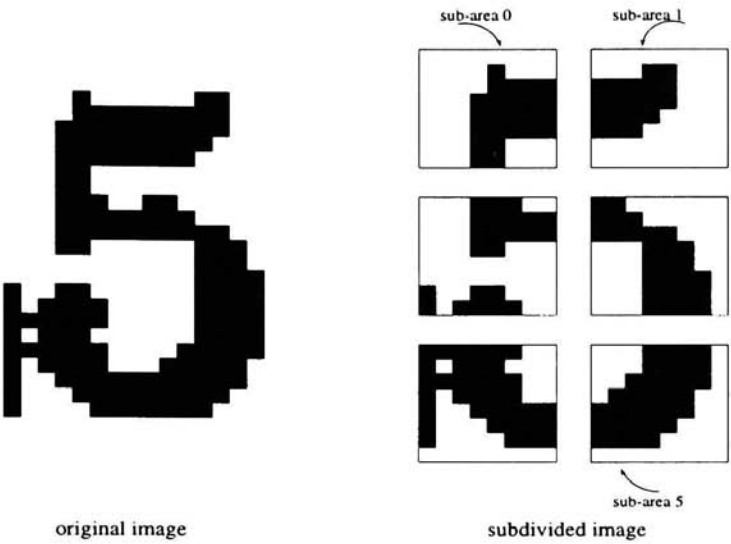


Figure 3: Division of the input in sub-images

the training set are situated near to the boundary. The boundary is defined by the addresses whose Hamming distances to both attractors are similar. In order to reduce the influence of the initial model structure in the modelling of the input distribution, the SOFT^d structure can be dynamically redefined during its training by eliminating and replicating nodes, according to the function they perform.

Figure 4 shows an example of how the attractors might divide a particular input space into two halves if connectivity 3 were used. A white circle is drawn for the input patterns that are nearer, in Hamming distance, to the attractor 0, and a black circle is drawn to represent the input patterns nearer, in Hamming distance, to the attractor 1. The larger the circle, the higher the frequency with which that particular input pattern occurs in the training set.

Immediately after the attractors for a particular node have been chosen, the value to be stored in each of its memory contents will be defined through competition between the attractors. This competition will determine which would be the node's most "natural" output. Once the architecture has been trained it can then be used to extract primitive features.

A good measure of the extent to which the pre-processed inputs generated by SOFT^d are likely to be more readily classifiable than the original input patterns, is provided by considering how SOFT^d affects the similarity, in terms

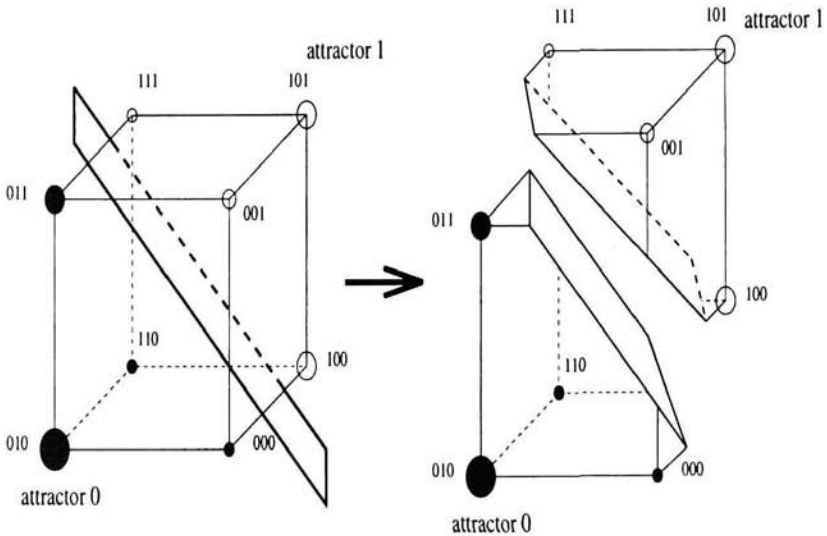


Figure 4: Attractors dividing the input space

of proportion of pixels with the same value, between patterns from the same class and between patterns distributed in different classes. The ideal situation is one where the similarity in the same class is kept high while the similarity among different classes is low.

Experimental evaluation shows that, using Hamming distance as a measure, SOFT^d achieves a useful data compression, reducing the pattern representations from 384 pixels to 96 pixels and increasing the average similarity between patterns from the same class and the average differences between patterns belonging to different classes, as can be seen from Table 1 and Table 2. While Table 1 illustrates the average similarities among patterns from the same class, Table 2 shows the similarities among patterns belonging to different classes.

According to the figures shown in Table 1 and Table 2, SOFT^d reduces the average similarity between patterns belonging to different classes and increases the average similarity between patterns from the same class. This should result in an improvement of the discrimination and generalization abilities of the Boolean classifier connected to SOFT^d. The following section illustrates the integration of SOFT^d and GSN^f in a fully integrated Boolean architecture.

Table 1: Similarities among patterns from the same class

Average pattern similarities (%)		
Class	*Original*	*Pre-processed*
0	86.1	88.6
1	83.6	85.7
1	85.7	92.7
3	74.8	75.0
4	76.2	78.6
5	78.9	81.9
6	83.5	92.0
7	86.1	94.2
8	79.0	87.6
9	73.8	82.7
Average	80.8	85.9
Deviation	4.8	6.3

4 Integrated Architecture

The design of modular neural architectures involves the division of the task in subtasks, the assignment of a neural module to each of the subtask and the communication among the modules.

As stated by[13], most of the approaches taken to implement modular neural networks are based on the *divide and conquer* philosophy, where the task to be solved is first divided in simpler, smaller tasks, which are then handled by different modules.

Regarding the communication among the modules, one of the main pre-requisites for a modular architecture is the compatibility of its internal inputs and outputs. In the SOFT^d-GSN^f modular architecture, this condition is satisfied, since the SOFT^d output is directly compatible with the GSN^f input. In the integrated architecture described in this chapter, the input image is first pre-processed by an already trained SOFT^d to be then directly used, without further pre-processing, as input by GSN^f.

SOFT^d and GSN^f are integrated by connecting the input terminals of GSN^f first layer nodes to the output terminals of SOFT^d last layer nodes, making them work like a single entity. Figure 5 illustrates the integrated architecture.

The training of the integrated architecture comprises two stages. In the first stage SOFT^d is trained using the original training set images as inputs.

Table 2: Similarities among patterns from the different classes

Average pattern similarities (%)		
Class	*Original*	*Pre-processed*
0	60.5	60.0
1	54.9	54.2
1	59.6	57.1
3	61.7	61.4
4	53.6	46.5
5	60.8	60.2
6	57.7	53.4
7	57.6	53.7
8	61.4	60.5
9	60.2	60.6
Average	58.8	56.8
Deviation	2.8	4.8

The second stage involves the training of GSN^f using as input the output provided by the propagation of the original input image through SOFT^d, using SOFT^d's recall phase. A pre-trained SOFT^d could also be linked to GSN^f. The recall phase of the integrated architecture is carried out by propagating the original input image first through SOFT^d and then through GSN^f using their respective recall functions.

For the purpose of evaluate the contribution of SOFT^d to the recognition performance achieved by GSN^f, both the basic GSN^f architecture and the modular architecture SOFT^d-GSN^f were trained to recognize numeral characters extracted from mail envelopes. In both cases, each GSN^f network uses three layers of nodes, where each node has connectivity 4.

The GSN^f networks were trained and tested with the original data and with the pre-processed data. In order to estimate, on a comparative basis, the effectiveness of SOFT^d pre-processing, the classification accuracy was measured after training the networks with three different learning algorithms previously used with the GSN^f classifier, which is to be used as the heart of the proposed modular architecture. These algorithms are the *Conventional lazy* algorithm (C^{lazy}) [14] , the *Deterministic lazy* algorithm (D^{lazy}) [14] and the *Progressive* algorithm (P) [15].

Table 3 shows the correct recognition performance achieved by GSN^f without pre-processing when different numbers of pyramids are used. Table 4

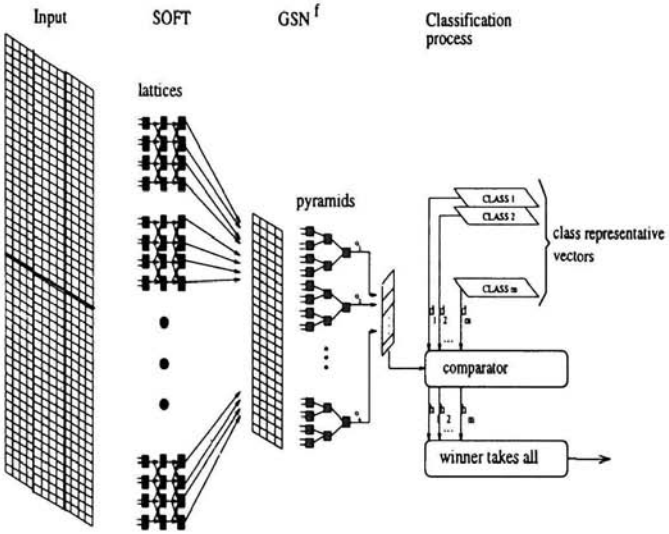


Figure 5: Integrated architecture

presents how the use of SOFT^d pre-processed data affects these results.

The results presented in Table 3 and Table 4 are the average of 10 different simulations. These results show that the use of pre-processed input leads to an overall improvement in the classification accuracy achieved by GSN^f.

It is interesting to notice that the smaller the number of pyramids used, the larger the difference in the correct recognition rates achieved by SOFT^d-GSN^f compared to GSN^f.

The larger difference between patterns from different classes and similarity between patterns from the same class has more importance for the recognition performance when fewer pyramids are used. This characteristic is particularly beneficial when hardware or time considerations limit the number of pyramids that can be used. It can also be seen that when 50 pyramids are used, GSN^f with pre-processed inputs accomplishes a recognition performance similar to that achieved by GSN^f with 100 pyramids using the original input patterns.

It is also clear from these results that a higher improvement in the classification accuracy occurs when GSN^f is trained with the C^{lazy} algorithm. The reason is that, due to a less ordered storing of values in the memory contents, the larger variety of pixels distributions in the original patterns affects more this algorithm than the others.

Table 3: GSN^f Classification accuracy

GSN^f classification accuracy			
Pyramids	*Learning strategy*		
	C^{lazy}	D^{lazy}	P
10	42.64	70.07	74.75
20	54.94	80.80	83.46
30	60.95	85.84	86.93
40	69.24	89.34	90.47
50	72.02	90.17	91.13
60	75.98	91.84	92.54
70	79.53	92.71	92.86
80	81.90	92.18	93.16
90	84.02	93.17	93.28
100	84.93	93.61	93.53

5 Conclusion

In this chapter, a modular Boolean architecture has been described on the basis of an integrated neural classifier. The proposed architecture uses an unsupervised learning algorithm where node blocks are trained in parallel and independently of each other as a front-end processor to generate pre-processed input to a GSN^f Boolean classifier.

Two sets of experiments evaluated the influence of the pre-processing provided by SOFT^d in the patterns similarities and the influence of SOFT^d on the correct classification rates achieved by GSN^f networks respectively. The first set has shown that SOFT^d increases the similarities among patterns from the same class and the diversity among patterns from different classes. The results achieved in the second set of experiments have demonstrated that the correct classification performance achieved by the GSN^f is improved when SOFTd is used as a front-end processor.

A. de Carvalho would like to acknowledge the support of the Brazilian Research Agencies CNPq and FAPESP

Table 4: SOFT^d-GSN^J Classification accuracy

SOFT^d-GSN^J classification accuracy			
Pyramids	*Learning strategy*		
	C^{lazy}	D^{lazy}	P
10	64.93	85.38	86.25
20	73.35	89.07	89.23
30	84.17	91.43	91.25
40	87.05	92.54	92.66
50	88.15	93.04	93.37
60	90.71	93.85	93.41
70	91.01	93.52	93.14
80	92.62	94.38	94.52
90	92.50	94.26	94.24
100	93.10	94.63	94.54

References

1. J. C. Houk. Learning in modular networks. In *Proceedings of the Seventh Yale Workshop on Adaptive and Learning Systems*, pages 80–84, 1992.
2. J. A. Fodor. *Modularity of Mind.* The MIT. Press, 1983.
3. A. Treisman. Features and objects in visual processing. In I. Rock, editor, *The Perceptual World*, page 200. Readings from Scientific American - W. H. Freeman and Company, 1990.
4. R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. HInton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 1991.
5. F. Z. Brill, D. E. Brown, and W. N. Martin. Fast genetic selection of features for neural network classifiers. *IEEE. Transactions on Neural Networks*, 3(2):324–328, March 1992.
6. H. Guo and S. B. Gelfand. Classification trees with neural network feature extraction. *IEEE. Transactions on Neural Networks*, 3(6):923–933, November 1992.
7. B. Nabet, R. B. Darling, and R. B. Pinter. Implementation of front-end processor neural networks. *Neural Networks*, 5:891–902, 1992.
8. M. Ishikawa. Learning of modular structured networks. *Artificial Intelligence*, 75:51–62, 1995.
9. A. de Carvalho. *Towards an Integrated Boolean Neural Network for Image Recognition.* PhD thesis, University of Kent, 1994.

10. A. de Carvalho, M. C. Fairhurst, and D. L. Bisset. Classifying images using goal seeking neural network architectures. *IEE. Proceedings-I Communications, Speech and Vision*, 40(1):12–18, February 1993.
11. E. Filho, D. L. Bisset, and M. C. Fairhurst. Architectures for goal-seeking neurons. *International Journal of Intelligent Systems*, 2:95–119, 1992.
12. E. Filho, M. C. Fairhurst, and D. L. Bisset. Adaptive pattern recognition using goal-seeking neurons. *Pattern Recognition Letters*, 12:131–138, March 1991.
13. R. Jenkins and B. Yuhas. A. simplified neural network solution through problem decomposition. *IEEE. Transactions on Neural Networks*, 4(4):718–722, 1993.
14. A. de Carvalho, M. C. Fairhurst, and D. L. Bisset. A. lazy learning approach to the training of GSN neural networks. In *Proceedings of the ICANN 92, Elsevier*, pages 673–676, Brighton, UK, September 1992.
15. A. de Carvalho, M. C. Fairhurst, and D. L. Bisset. Progressive learning algorithm for GSN feedforward neural architectures. *Electronics Letters*, 30(6):506–507, March 1994.

DETECTING DANGER LABELS WITH RAM BASED NEURAL NETWORKS

C. LINNEBERG, A.W. ANDERSEN, T.M. JØRGENSEN and S.S. CHRISTENSEN
Risø National Laboratory, P.O. Box 49,
DK-4000 Roskilde, Denmark

An image processing system that automatically locates danger labels on the rear of containers is presented. The system uses RAM based neural networks to locate and classify labels after a pre-processing step involving non-linear filtering and RGB to HSV conversion. Results on images recorded at the container terminal in Esbjerg are presented.

1 Introduction

The problem of locating danger labels on an unknown background is addressed. For the case considered the background consists of different container rears. The background can have different colours and texturing patterns. One container can have from zero to around seven different labels attached to its rear. There exist 25 different types of labels (see figure 1).

Due to wear and tear caused by weather and age a specific type of label will appear slightly different from one rear to another (the colour tone might vary; small parts of the label might have been worn off; parts of the label might be partially covered by dirt). Also the illumination and weather conditions at the time of inspection will vary. Accordingly the detection scheme to be used must be very robust. It is possible to assume that only minor rotations and scaling of the labels will occur as labels by regulation must be oriented in a specific way and have a predefined size.

A solution of the described task provides the possibility of automatically inspecting the labelling of container goods. We present a solution based on a RAM based neural network[1,2] and a controllable camera. The work has been accomplished in collaboration with an industrial partner.

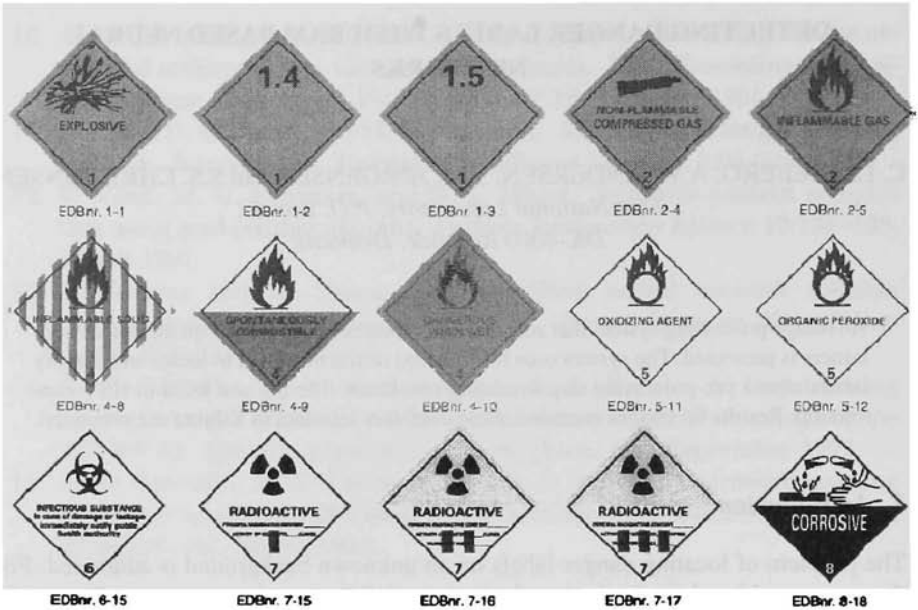


Figure 1: Examples of the different types of danger labels that must be classified. The different categories are distinguished by their colour. The major colours used are red, green, yellow, and blue.

2 Concept

The inspection of a container is divided into the following steps:

- Locate possible label objects based on the shape of the object.
- Move the camera fovea to obtain a high resolution image of the object.
- Use the colour and pictogram information of the objects to perform a classification.

Both when locating the labels and when performing the final classification the image data are initially pre-processed to create a binary pattern. These patterns are fed into RAM based neural networks for classification.

It is of course essential that regions with labels are actually detected. Accepting some false positive classifications reduces the chance of a false negative. Within the

first task it is therefore acceptable that a few label-free regions are classified as regions of interest. The following classification step must then detect that no labels are actually present.

The first step has been tested on real-world images while the latter part is still a laboratory set-up.

3 Localisation of the labels

Initially the camera is zoomed out to record a full image of the container rear. The next step is then to find objects with label-like shapes. This is done by scanning the image with a search window. The dimensions of the window are slightly larger than the dimensions of the labels. For each window position a RAM based system is used to determine whether a label shaped object is present within the window.

A pre-processing of the data within the search window is required since the RAM based system requires binary input. The labels are characterised by edges oriented at 45° and 135° with respect to the horizontal line. Non-linear edge detection filters were developed to detect the contour of a label.

3.1 Edge detection

Filter kernels designed to detect tilted edges of 45° and 135° were used. The filter result is a binary image where each pixel classified as an edge pixel is assigned the value 1. The operation of the filter is based on the hypothesis that a pixel can be considered an edge pixel. The average intensities are calculated for the background and object. The hypothesis is now tested using these two average intensities and their corresponding variances. For further description of this non-linear edge detection filter see Jørgensen et al.[3]. Figure 2 illustrates how this filtering scheme extracts the edge information and creates a binary example from an image of a label. In order to locate the striped label (see figure 1), a vertical edge detecting filter was added to the pre-processing scheme.

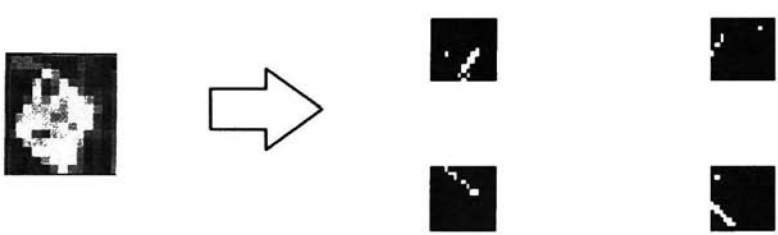


Figure 2: This example shows how a label is converted to a binary pattern. Four line detectors are used. For each orientation separate detectors are used to detect light to dark and dark to light transitions.

3.2 Training of RAM based neural network

The output of the edge filters is fed into a RAM based NN system. The purpose of the NN is to mark regions of interest in the image. A search window is centred on examples of labels and the system is trained to detect the presence of a label.

The translation invariance needed to make the scheme robust is partially obtained during the filter step and partially by assigning different classes to different positions. For each label used for training a range of examples is produced. Each of these examples corresponds to a specific (small) shift within the search window. Each shift is assigned a specific class to avoid interference. Producing these different classes causes the system to become invariant to small shifts of the labels' position within the search window. Consequently the container rear can be scanned with a larger step size. Dark labels on light background and vice verse as well as the striped label are treated as separate classes in order to simplify the classification task for the neural network.

The network also needs examples of label free regions. The examples to be used must correspond to regions having features that can be confused with the ones desired.

A way of obtaining such examples is the following:

1. Store a range of label examples in the RAM net together with an example of a totally empty region.

2. Apply the RAM net on a number of container rears (2-5).

3. Use the false positive detected regions as examples of label free regions.

If necessary step 2 and 3 are repeated a couple of times.

4 Camera set-up

The image used to locate potential danger labels do not provide sufficient resolution for proper identification of the danger labels found. In order to overcome this limitation an active vision system was set up. The use of a motorised camera with controllable pan, tilt and zoom makes it possible for the system to zoom in on a region of interest. One can then obtain a high resolution image of the potential label and thereby gain the information needed to perform a final identification of the object. The camera used is a standard surveillance camera capable of 170° pan, 110° tilt and zoom from 4.5° to 38°. The camera controller developed is running as a separate task allowing the localisation and identification system to perform further processing while the camera is moving.

5 Label Identification

The RAM based neural network produced by the training procedure outlined in section 3.2 outputs two votes, numbers corresponding to a label being present and a label-free region, respectively. Using a Winner-Takes-All decision the votes are used to mark regions of interest in the image, i.e. regions that are likely to contain a label. Having detected the regions of interest the next step is to verify and classify the potential label. First the camera is zoomed in onto the object to obtain a high resolution image. At this stage the edge detection filters are used to re-locate the label.

Figure 3: Edges detected in close-up image of a danger label.

From the detected edge pixels (see figure 3) we calculate the equations for the four line segments forming the contour of the label. An example of the result of this can be seen in figure 4.

Danger labels are colour coded. Accordingly it seems sensible to use this colour information for the classification of the labels. A way of handling colour information is to use the hue, saturation and intensity representation[4]. For the upper and lower halves of the label a hue histogram is calculated. The colour of the upper and lower half of the label will give the main class of the label. In the cases where the colour information does not give sufficient information the pictogram in the upper half of the label is classified by another neural network.

Figure 4: High resolution image of a potential danger label. The rectangle indicates the position of the label (calculated at lower resolution, hence the offset).

5.1 Colour histogram

To obtain the colour of the upper and a lower half of the label a hue histogram is c culated. The hue circle is divided into the overlapping colours regions red, orange, blue as well as a special case for no colour defined as pixels with a low saturatior being nearly black or white. See figure 5.

For several of the labels this information will be sufficient to perform a segm tation into different main groups. The remaining information is provided by the pic gram in the upper half of the label.

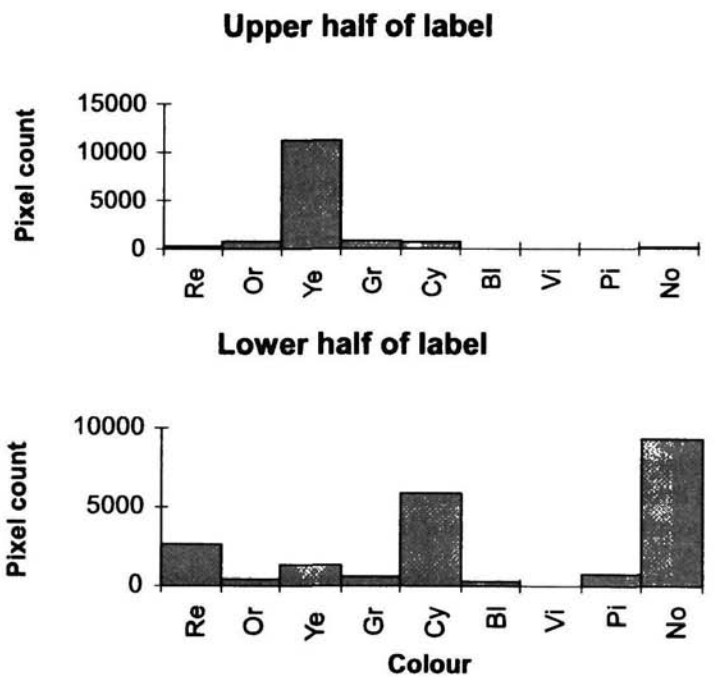


Figure 5: Hue histogram for the upper and lower half of the label. The colours are Re(d), Or(ange), Ye(llow), Gr(een), Cy(an), Bl(ue), Vi(olet), Pi(nk), and the special case "No colour"

5.2 Pictogram recognition

Whenever the colour data does not contain sufficient information to perform a identification of the danger label, classification of the pictogram at the top of the la will provide the missing information. For a few types of labels ambiguity will still ist e.g. "poison" versus "poison gas" or "inflammable gas" versus "inflammable l

uid." A solution to this ambiguity could be to apply an OCR reader to the text line in the middle of the label and/or to the number at the bottom of the label.

From the high resolution image of the label we cut a square region containing the pictogram (see figure 6). The pictograms are pre-processed and passed to a RAM based neural network for classification. The pictogram image is divided into local areas for each of which a grey scale histogram is calculated. These histograms are reduced to only 3 grey levels in order to reduce the input space to the neural classifier. Translation invariance is obtained by the use of local histograms and by variation in the training set.

Figure 6: Examples of pictograms cut from the danger labels.

Figure 7: Example of a container rear end containing 4 labels. The white squares indicate the regions indentified by the sustem as regions containing labels. All four labels are correctly located.

6 Results

In order to obtain realistic images for training and testing purposes a video camera was installed at the DFDS container terminal in Esbjerg. Each time a container entered the terminal a video sequence of 10 seconds was recorded onto a VHS tape. These video sequences were subsequently digitised for further processing.

The processing time on a 486-66 MHz machine for one container rear is around 10 seconds for the label localisation. An example of how the algorithm successfully locates several labels on a container rear without performing any errors is shown in figure 7. The active vision and label identification part of this project is still to be tested in a real world environment, but from our laboratory tests this scheme appears to be robust.

7 Conclusion

We have presented a system for automatic localisation and classification of danger labels on the rear of containers. The system uses a combination of traditional image processing methods and RAM based neural networks. For the label localisation system the test results were obtained on images recorded under real conditions.

Acknowledgments

Part of this work was funded by the EUREKA project HERA. The partners were Rambøll, Hannemann & Højlund (DK), Applied Bio Cybernetics (DK), BICC (UK) and Risø (DK).

References

1. I. Alexander and T.J. Stonham, *Computers and Digital Techniques*, **2**, 29-40 (1979).
2. I. Alexander, An Introduction to Neural Computing (Chapman and Hall, London, 1990).
3. T.M. Jørgensen, S.S. Christensen and A.W.Andersen, *Pattern Recognition Letters*, **17**, 399-412 (1996).
4. Finlayson, Colour object recognition, (Simon Fraser University, School of Computing Science).

FAST SIMULATION OF A BINARY NEURAL NETWORK ON A MESSAGE PASSING PARALLEL COMPUTER

T. MACEK
Department of Computers, Faculty of Electrical Engineering,
Czech Technical University, Karlovo namesti 13,
121 35 Praha 2, Czech Republic

G. MORGAN, J. AUSTIN
Advanced Computer Architecture Group, Department of Computer Science,
University of York,
York Y01 5DD, United Kingdom

This text proposes an efficient method for the implementation of the ADAM binary neural network on a message passing parallel computer. Performance examples obtained by testing an implementation on a Transputer based system are presented. It is shown that simulation of an ADAM network can be efficiently sped-up by parallel computation. It is also shown how the overlapping of the computation and communication positively influences performance issues.

1 Introduction

The speed of simulation of Artificial Neural Networks is one of the crucial problems slowing their more widespread application. Conventional workstations are not fast enough and special purpose parallel hardware is often unacceptable on cost grounds. The alternative solution, of using a general purpose parallel computer, is more cost effective, leads to a shorter development time and results in a more maintainable system.

Most neural networks have long teaching and recall times due the use of evaluation functions which are slow to compute. For example, weights based upon floating point values result in the use of slow floating point computations. Binary neural networks (where the weight is only either 0 or 1) rely upon much faster logical operations. Therefore teaching and recall in such neural networks is much faster. However, much larger networks are also used in many applications. Therefore we have focused on the development of methods for the simulation of binary weighted neural networks on parallel computers.

In this text we describe our experience in implementing the ADAM neural network on a message passing computer system. We present the techniques used in our implementation as well as some of the results obtained by running experiments on the Transputer based Meiko machine with 32 processors.

The first results of this work were presented in[1]. We focus here on the im-

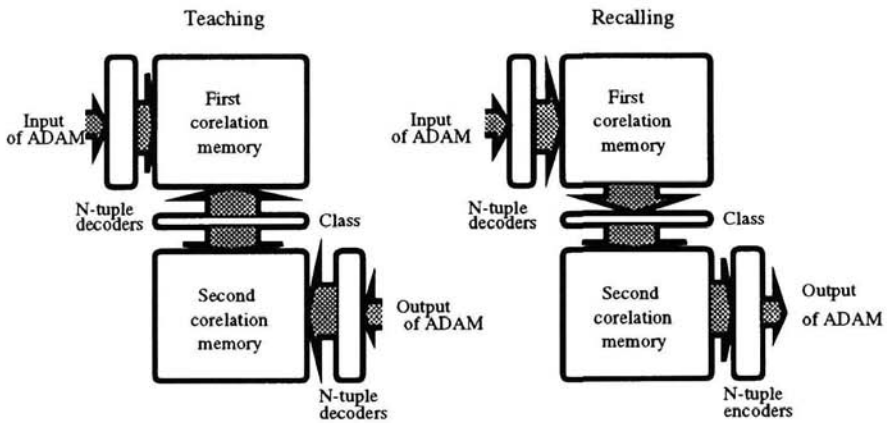


Figure 1: The structure of ADAM neural network.

provements which have been obtained by a higher level of the parallelism in the communication and by overlapping the computation and the communication.

In the next section we briefly describe the structure of an ADAM network. The third section is devoted to the issues which we considered important in the design of the parallel implementation. This is followed by the fourth section which is devoted to the presentation of results. The final section of the text is devoted to a summary of the results.

2 ADAM

In this section we will describe very briefly the ADAM system (see [2] for more details). ADAM stands for Advanced Distributed Associative Memory. It is able to store and retrieve associations between pairs of the binary vectors, even if the vectors are incomplete or corrupted. The structure of ADAM is depicted in figure 1.

Before use an ADAM is taught the associations required of it. This is called Teaching. When in use an ADAM system is used for Recalling the data taught to it. We now describe teaching and recalling in more detail.

During *teaching* the association between binary vectors applied to the inputs and outputs is stored. The structure of ADAM is based on the interconnection of two binary correlation memories. The input vector (after n-tuple preprocessing see [3]) is associated with so called class vector in the first correlation memory. The output vector (after n-tuple preprocessing) is associated

with the same class vector in the second correlation memory.

During *recalling* the input binary vector is applied to the inputs of the ADAM. After n-tuple preprocessing the associated class vector is retrieved from the first correlation memory. The class vector is consequently used for retrieving the association from the second correlation memory. The output of ADAM is obtained by n-tuple encoding of the resulting vector.

All class vectors have a fixed number of the elements set to one.

During n-tuple preprocessing the binary vector which forms the input to the system is divided into smaller vectors which are applied to the inputs of the N-tuple decoders. Each decoder has only one output set to one. This results in a longer but much more sparse vector to be associated in the correlation memory.

The correlation memory uses a binary matrix for storing information. Initially it is set to zero. During teaching input vectors are applied to the rows and output vectors to the columns. Those bits in the binary matrix are set where corresponding elements in a column and a row have the value one. The other bits remain the unaltered.

During recalling an input vector is applied to the inputs. The sum is calculated of all the places where both matrix element and corresponding element of input vector have the value one (for each of the columns separately). The resulting vector is then thresholded to obtain a binary vector.

It can be seen that ADAM depends mainly upon Boolean operations. As these are fast on computers so also is ADAM.

3 Parallelisation Issues

This section is devoted to the description of the issues we considered important in the design of the parallel implementation of ADAM.

In the first part of this section we will describe the methods which are generaly used for the parallel implementation of the neural networks. We will follow by discussion of the technique of mapping of ADAM on the processors and the topology of the neural network. Finally we will describe the system of communication and synchronisation both inside and outside of the processors.

Two methods of the parallel implementation of the neural networks are commonly used.

The first method is based on distribution of the training and testing set. Each processor simulates the whole neural network but with a different data. The problem of this method is how to keep the consistency of several copies of the neural networks on a different processors. It is a task which is very time consuming particulary for binary neural networks.

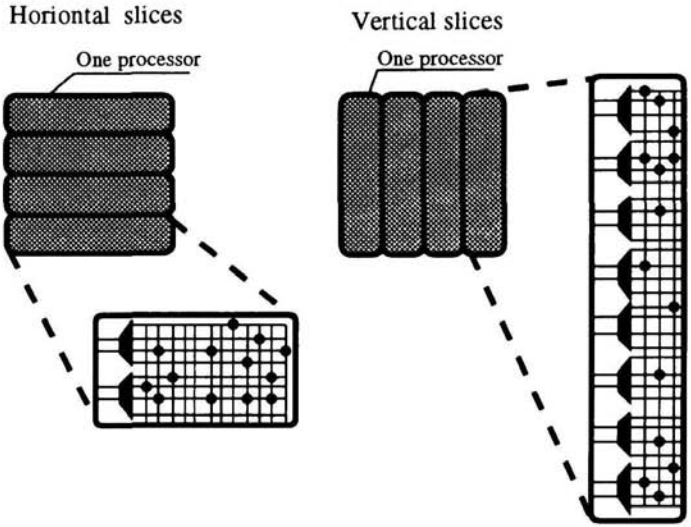


Figure 2: Candidate methods for the distribution of the correlation matrixes.

The second method is based on distribution of the neural network itself. Each processor is responsible for a part of the neural network. Processing of each datum is done in parallel. This method is less memory consuming, provides speed-up of the processing of even single datum and has no troubles with keeping the consistency of data on a different processors. This is also the method which we used in our experiments.

In order to determine which would be the most effective parts of ADAM to parallelise we profiled several networks and observed where the timing bottlenecks were. These were found to be the teaching and recalling of associations from the correlation matrixes. The construction of the tuples and the class vector generation were much faster. Therefore, we decided only to parallelise the operations of the correlation matrices.

Figure 2 shows the candidate methods for the distribution of the correlation matrixes betweend processors. The left part of the figure shows the method which we call *horizontal slicing*, the right part of the figure shows the *vertical slicing method.* We argue that horizontal slicing method is superior to vertical slicing method. It is because if the horizontal slicing method is used, the computation load can be better balanced between processors. The problem with the vertical slicing method is that each processor uses just part of the class

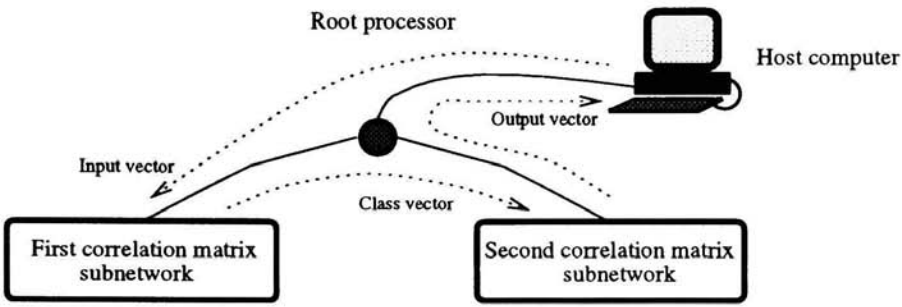


Figure 3: The separate processing of the correlation memories.

vector. Each part of the class vector can include different a number of bits set to one and moreover this number changes with the class. If the vertical slicing method is used, whole class vectors are processed at each of the processors. By altering the number of the tuples we can adjust the computation load at the processor.

The two correlation matrices are stored and executed in two separate branches of a tree, as shown in figure 3. The root of the tree is a processor which receives the system's input and transmits it to the subnetwork handling the first correlation matrix. When this subnetwork has finished working on its result (the class vector), it is returned to the root which then communicates it to the second correlation matrix. The recalled pattern from the second CM is then returned to the root transputer for output from the system.

In the case of teaching the two CMs can be evaluated in parallel, in the case of the recalling pipelining can be used. For these reasons it is better to allocate an independent subnetwork for processing each correlation matrix. The number of the processors in each subnetwork is determined according to the number of tuples in the CM concerned. Therefore the two CM networks need not be the same size.

Figure 4 shows the proposed topology of the networks of the whole system. The structure is based on interconnection of two ternary trees. The choice of the topology of subnetworks was determined by the type of the communications required. It is not necessary to communicate between any arbitrary pair of the processors but the operations for broadcasting, scattering the data to all transputers and gathering the results must be implemented efficiently. For this purpose a star or a tree are the optimal structures. We used a ternary tree because of the number of links on the transputers.

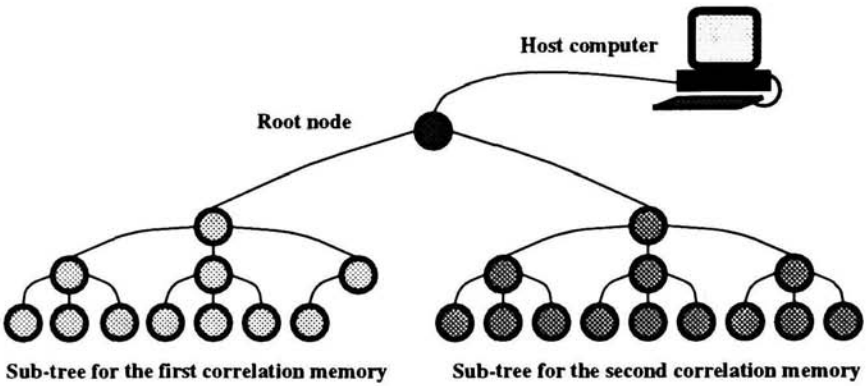


Figure 4: The topology of the network of transputers.

Command	Function
RECALL_CM	recall operation for one input pattern.
TEACH_CM	teach one association.
WRITE_M_CM	distribute correlation matrix to the processors.
READ_M_CM	collect matrix from the processors.
MAKE_MIXED_CM	create ADAM in the network.
INIT_TREE_CM	initiate particular processor.
EXIT_CM	finish processing of whole network.
READ_VAR_CM	reading of the internal variables (for debugging).

Table 1: Table of commands used for communication in the trees.

All processors in the CM subtrees behive similarly since they are driven by the same program. It is based upon a command interpreter loop consisting of reading a command from the parent processor and executing the corresponding operation. The set of the commands used is shown in table 1. By way of an example we will describe the commands for software initialisation of the network and for the recall operation.

The commands for software initialisation of the network are the first which should be sent to the CM subtrees. The command INIT_TREE_CM (see table 1) is sent to the each processor separately. This is followed by the address of the processor and the parameters of that slice of the CM, including the number of tuples to be allocated at that particular processor.

When a processor receives a command it checks the address field. If the

configuration command is not addressed to that processor it forwards the command and data to the next layer. After sending the initialisation command to each processor it has information about which part of ADAM is to be processed at that processor as well as information about the number of tuples in all its subtrees. This is necessary for fast distribution of the data in the processes of teaching and recalling.

The configuration process described above makes it possible to allocate different numbers of tuples to different procesors. Therefore the configuration procedure can be used for better load balancing.

The recall command (INIT_TREE_CM; see table 1) is followed by block of inputs to the tuples of the whole subtree. The processor receiving this data has knowledge about the length of the block obtained from the process of configuration (it is equal to the sum of the tuples in all subtrees plus the number of tuples to be processed at that processor). After receiving the block of data from the previous layer, the recall command is broadcast to the subtrees of the processor (if there are any) followed by the parts of the received data belonging to the subtree only. The rest of the data are inputs for the parts of the ADAM allocated on that particular processor. Therefore the amount of data transmitted is high near to the root of the tree, but decreases as the data goes down the tree. Recalling the part of the ADAM allocated to a particular processor is performed and the processor waits for the results from the subtrees (if there are any). The results from the subtrees (which form the class vector and are all of the same length) are added to the local results and sent to the previous level.

A processor can run one or many processes. However, the overhead related to switching between processes increases with the number of the processes. Therefore we attempted to keep the number of processes on a processor as small as possible. The reason why we used more processes at the same processor was to take the advantage of parallel computation and communication. We used two processes for each of the links (one for transmitting, a second for receiving) and one process for computation. We will show in the next section that this system improved the speed of the simulation significantly compared with our first version of the program (see [1]) where just one process was allocated at each processor handling both communications and computation.

The data transfer and synchronisation between processors is based on communication through channels. Inside of the processor we used shared memory for data sharing and semaphores for synchronisation of the processes.

Figure 5 illustrates, by means of a Petri net [4], an example of synchronisation between processes. The example is for recalling. The graph consist of two types of nodes: *Places*, and *Transitions*, with *Places*, drawn as circles,

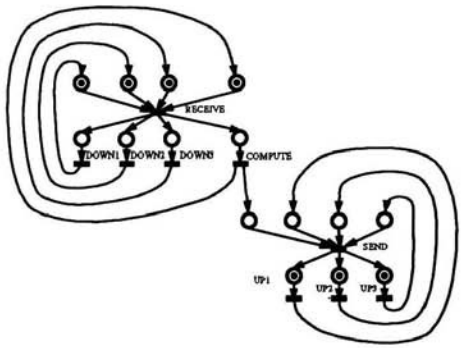


Figure 5: Synchronisation of processes.

representing conditions, and *Transitions*, drawn as bars, representing events. *Tokens*, represented as black dots within a place, symbolize the state of the net. There can be a finite number of tokens at a place. The transition is enabled if all of its input places contain at least one token. When a transition occurs one token is removed from all of the input places. When a transition finishes a single token is added to each of the output places. In our case a transition is the processing of one datum by one process.

There are four types of transitions in the above diagram. The *RECEIVE* transition receives input data from the parent layer of the tree. A subset of that data is then distributed to the subtrees by the transitions *DOWN1, DOWN2, DOWN3.* The rest of the data are used by the transition *COMPUTE* for local recalling. The results from the subtrees are received by the transition *UP1, UP2, UP3.* The final result, which is obtained by summation of the local results with the results from the subtrees, is sent to the parent layer by the transition *SEND.*

All transitions can be activated in parallel. While *COMPUTE* works with its part of the data and the rest is sent to the subtrees (*DOWN*), the data for next computation can be received (*RECEIVE*) and the results from the previous computation can be sent to the parent layers (*SEND*). In order to do this there are input and output buffers on the processors.

The Petri net in figure 5 is initialised by the existence of tokens in the places enabling the *RECEIVE* and *UP1, UP2, UP3* transitions. The number of the tokens at each of those places corresponds to the length of the buffer.

During processing the *RECEIVE* transition is enabled if there is still place in the input buffer. If there is at least one datum in the input buffer it can be processed by the *COMPUTE* and *DOWN1, DOWN2, DOWN3* transitions.

The smallest number of the tokens in one of the input places of *RECEIVE* represents the length of the empty part of the buffer. Similarly the smallest number of tokens in the input places of the *SEND* transitions represents the the number of the data which should be sent to the previous layer. The *SEND* is enabled if the results have been received from the subtrees (*UP1, UP2, UP3*) and the local computation (*COMPUTE*) has also finished. The *DOWN* transitions are enabled if there is at least one item in the input buffer. The *UP* transitions are enabled if there is at least one empty place in the output buffer.

If a processor is the leaf of the tree or if it has less than three subtrees, the corresponding processes *DOWN* and *UP* are not placed at the processor.

Implementation of the teaching is similar, but the additional processes *UP* and *SEND* are not used.

In this section we have described the main elements of our parallel implementation. We now present some results illustrating how well the implementation works.

4 Evaluation of results

This section presents some of the results obtained by running experiments on a Meiko system.

The examples have been selected so that both best and worst case performance will be shown.

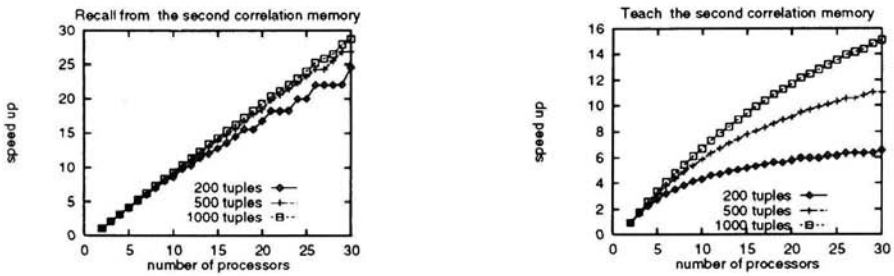
The Meiko system has thirty two T800 Transputers each with 4MB memory. Software was written in the C language, using the low level channel based communication between processors.

The speed of the simulation of the first and second correlation memories was evaluated separately. The vectors used for teaching and recalling were generated randomly as were the weights in the binary matrixes. Each measurement was performed for a block of 50 data, however the results are the average values calculated for one datum.

The teaching operation is significantly faster. The average amount of time spent in teaching the first correlation matrix of an ADAM (1000 tuples, 4-bit tuple, 32 byte class size) on one processor is 186.4μs. The average time for recalling a correlation memory of the same size is 1707.2μs.

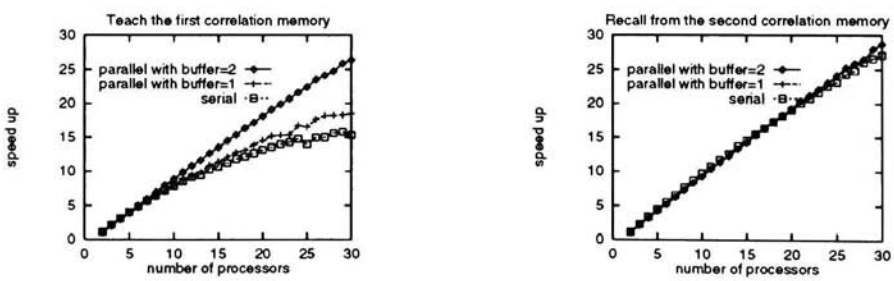
Figure 6 shows how speed-up depends on number of processors. The speed-up has been obtained by dividing the time taken to run the parallel version of ADAM with the time to run a uniprocessor version.

The results for larger ADAMs and for recalling are better. This is because the computation load is relatively higher than communication load.



ADAM size: 4-bit tuple, 32 bytes class size.

Figure 6: The speed-up of obtained for various numbers of processors.



ADAM size: 1000 tuples, 4-bit tuple, 32 bytes class size

Figure 7: Comparison of the efficiency of three methods of the implementation of ADAM.

Figure 7 illustrates how the speed-up depends upon the technique used in the implementation. Three techniques are compared. The first curve (marked *serial*) correspond to our first version of the program where just one process was running on each processor. The second (marked *parallel with buffer=1*) used the method described above with communication and computation in parallel but with no buffering. This means that data were distributed down the tree in parallel with computation but the next datum was not received in the same time from the previous layer. The third curve (marked *parallel with buffer=2*) corresponds to the fully parallel version described in this text.

It can be seen that the performance has been improved significantly by parallel computation and communication.

5 Conclusions

We have briefly presented here a method for the parallel simulation of the ADAM neural network. It has been shown that the simulation of the ADAM neural network can be efficiently sped-up by parallel computation.

Acknowledgements

We would like to express our thanks to the British Council and Amstrad, which have been sponsoring this research in the framework of the Excalibur Scheme. We would also like to thank to our employers, The Department of Computer Science, The University of York and The Department of Computers at FEL, CVUT in Prague who made it possible to do this work.

References

1. T. Macek, G. Morgan, and J. Austin. *A transputer implementation of the ADAM neural network.* In *World Transputer Congress'95*, (1995).
2. J. Austin and T.J. Stonham. *The ADAM associative memory.* YCS report YCS94, University of York, Department of Computer Science, Heslington, York, YO1 5DD, Englan, (1987).
3. W. W. Bledsoe and I. Browning. *Pattern recognition and reading by machine.* In *Proceedings of the Eastern Joint Computer Conference*, (1959).
4. E. V. Krishnamurthy. *Paralel Processing Principles and Practice.* Addison-Wesley Publishers, (1989).
5. D. J. Willshaw, O. P. Buneman, and H. C. Longuet-Higgins. *Non-Holographic Associative Memory. Nature 222*, pages 960–962, (1969).

6. J. Austin and S. Buckle. *The practical application of binary neural networks.* In *UNICOM seminar on Adaptive computing and information processing*, (1994).
7. G. Smith and J. Austin. *Analysing aeiral photographs with ADAM.* In *Proceedings of the UNICOM seminar on Adaptive computing and information processing*, (1992).
8. T. Macek and M. Snorek. *Neural net on transputers.* In *Progress in Transputer Computing Technology TCT'93*, (1993).
9. F. Meunier and P. Zemanek. *Concurrent algorithms for transputer networks.* In *SOFSEM'92*, (1992).
10. J. Kennedy, J. Austin, and B. Cass. *A hardware implementation of a binary neural image processor.* In *IEE conference on Image Processing and its Applications 95*, (1995).

C-NNAP: A DEDICATED PROCESSOR FOR BINARY NEURAL NETWORKS

J.V. KENNEDY, J. AUSTIN, R. PACK, B. CASS
Advanced Computer Architectures Group, Department of Computer Science, University of York, Heslington, York, YO1 5DD, UK

This chapter describes techniques for the hardware implementation of a Correlation Matrix Memory (CMM), a fundamental element of a binary neural network. The training and recall of a CMM based system is explained, prior to the hardware description of the dedicated processing platform for binary neural networks, C-NNAP. The C-NNAP architecture provides processing rates nearly eight times faster than a modern 64-bit workstation. It hosts a dedicated FPGA processor that performs the recall operation. The data flow through a multiple board system is also described, which will provide an even more powerful processing platform.

1 Introduction

The primary focus of neural network research has been into real-valued networks: networks whose connections have weights that are represented using floating point numbers. Unfortunately, this does cause problems as a floating point calculation usually requires multiple clock cycles, resulting in a time consuming training and recall operation. This is exacerbated in some research domains such as image processing systems which often have input data with high dimensionality. To overcome some of the problems of real-valued networks, an area of active research is the field of binary neural networks, sometimes described as *Weightless Neural Networks*[16]. Binary networks have the advantage of single pass training and recall using boolean operations. There is extensive ongoing research into the theory[13], implementation and application of binary networks. The uses of binary neural networks range from address database deduplication[11], rule-based systems[5], probability density estimations, time-series analysis of financial data[9], to molecule matching[14]. For problems of a reasonable size, such as an input image of 512^2 pixels, or a 25 million address database, there is a significant amount of data to be processed. To enable this data to be processed in real-time, e.g. camera frame rates for image work, then it is cost effective to use dedicated accelerating hardware. The hardware that the group has developed and is currently using is the Cellular Neural Network Associative Processor, C-NNAP.

Section 1 of this chapter describes the operations required to train and recall from a binary neural network. Section 2 describes the hardware design of the C-NNAP ar-

chitecture. An element of the C-NNAP design is a dedicated FPGA processor, the SAT, which is described in detail in Section 3. Section 4 demonstrates the performance advantages of the architecture and provides equations for users to calculate processing rates on their data sets. The chapter concludes in Section 5.

2 Binary Neural Networks

A binary neural network contains a binary matrix, often called a Correlation Matrix Memory (CMM), that stores relationships between two binary vectors. They are advantageous as they can be taught or tested in a single pass over the data using boolean operations, this results in simple and fast processing. The networks do have drawbacks, they may not generalise as well as a MLP (Multi Layer Perceptron) network, nor can they solve as many problems as an MLP. However, Morciniec[12] stated that "*RAMnets are often capable of delivering results which are competitive with those obtained by more sophisticated, computationally expensive models.*" These models included back propagation and radial-basis functions, which require many hours of training time. There are therefore many situations where a slightly reduced functionality is acceptable as the processing time is of primary importance.

Historically, binary neural networks originate from work done in the 1950's, such as that done on the N-tuple system. The N-tuple system was developed by Bledsoe and Browning[6] for the recognition of alphanumeric patterns, and popularised by Aleksander[2]. Bledsoe and Browning's system had a binary matrix for storing data, and used the N-tuple system for pre-processing the image data. Further work was done on the binary matrix system by Willshaw[15]. Willshaw's thresholding mechanism however, is not tolerant of noise in the input date. In this scenario it is better to use *L-max* thresholding which is used in the two-layer CMM network of the Advanced Distributed Associative Memory, or ADAM[3]. The ADAM network was specifically designed for use in scene analysis.

The interesting feature of the above networks is that they are a superset of the basic correlation matrix memory (CMM). The training and recall from a CMM is now explained.

2.1 Training of the CMM Network

Training a CMM network requires two binary vectors. The first vector, known as the training pattern, is the output from the pre-processing stage. The pre-processing could be grey-scale n-tupling[4], CMAC conversion[1], or any technique that converts an input to a binary vector. The second binary vector required is the output pattern, and is known as the separator pattern (sometimes known as the class pattern). The separator patterns are as near to mutually orthogonal as possible which reduces clustering of data in the network, this in turn reduces the post-processing required.

A mathematical description of the training phase is shown in Equation 1 where p_k is pre-processed input vector k, s_k is the separator pattern to be associated with the input vector. The symbol $\oplus$ indicates a bitwise OR, M is the binary matrix, and N is the input data set. Using the transpose of the separator pattern vector and the input pattern vector, an outer product operation (or correlation) is performed. This results in a binary matrix as shown in Figure 1 where a binary "1" is represented by a circle, a "0" by an absence of a circle. Further input patterns undergo the same procedure and are then superimposed on top of the original matrix using the bitwise OR.

$$\begin{cases} M_o = 0 \\ M_k = M_{k-1} \oplus s_k^T p_k, k \in N \end{cases} \tag{1}$$

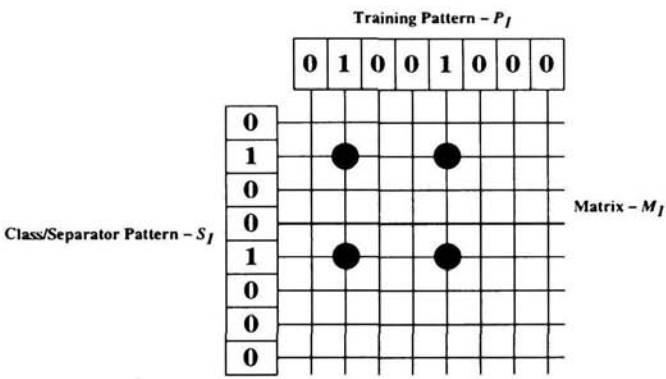


Figure 1: CMM Training Phase

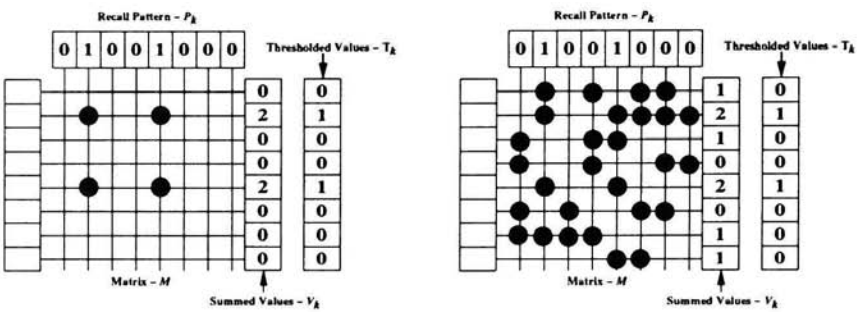
2.2 *Recall from the Network*

Recall from the network is described by Equation 2. This is an inner product operation, where p_k is pre-processed input vector k, after undergoing a transpose. M is the binary matrix. The output from this operation is a vector of integer values v.

$$v = \sum_{i=1}^{n} (M p_{ki}^T) \tag{2}$$

In architectural terms, the inner product means that each input `1' activates a col-

umn of weights in the matrix. These columns are then *integer accumulated*, resulting in the summed values of vector, v. This is shown in Figure 2(a) for a network containing a single correlation. Figure 2(b) shows the recall from a network that has been taught many correlations.



(a) ADAM Recall - Sparse Network

(b) ADAM Recall - Saturated Network

Figure 2:

The vector of summed values, v_k, is thresholded using either *L-max* thresholding or Willshaw thresholding. *L-max* thresholding sets the L highest summed values to "1", all other summed values are set to "0". Willshaw thresholding is described by Equation 3. This sets the summed values in vector v that equal W to "1", where W is the number of bits set in the input pattern, the remaining summed values are set to "0". An often used variant of Willshaw thresholding sets all values in vector v_i that are greater than or equal to w to 1, all others are set to 0 (this provides for better control of noisy input data). The output from thresholding vector t is the original separator pattern.

$$t = \begin{cases} v_{i=W} = 1 \\ v_{i \neq W} = 0 \end{cases} \tag{3}$$

It should be noted that the thresholded vector may actually contain more than one separator pattern and that further processing would then be required to extract the individual patterns. The technique used is called Middle Bit Indexing (MBI), and is described in[7] and in greater detail in [10].

3 The Cellular Neural Network Associative Processor

In many binary neural network applications, the training and recall data needs to be processed at fast rates. For image work, camera frame-rates sometimes need to be maintained (typically 25 frames a second in the UK). To process 25 million addresses in a day also requires a processing rate of at least 17,400 addresses a minute. In order to make this possible it is necessary to use dedicated hardware: the C-NNAP platform. C-NNAP is a dedicated platform for processing binary neural networks, and is available to all users on the department network.

C-NNAP is based on a standard VME bus system which makes the architecture available to a wide range of computer platforms. The system's host-computer, the S-Node, is a Silicon Graphics Challenge DM server with an integral VME bus. The host computer is required to pre-process the data and control data movements within the system. The VME bus of the S-node hosts the C-NNAP nodes, or C-nodes, which are tasked with the training and recall of the CMMs. A large CMM can be distributed over multiple C-nodes, on completion of C-node processing the S-node combines the results from all the boards. The architecture of a C-node is shown in Figure 3, and a photograph of a completed board is shown in Figure 4.

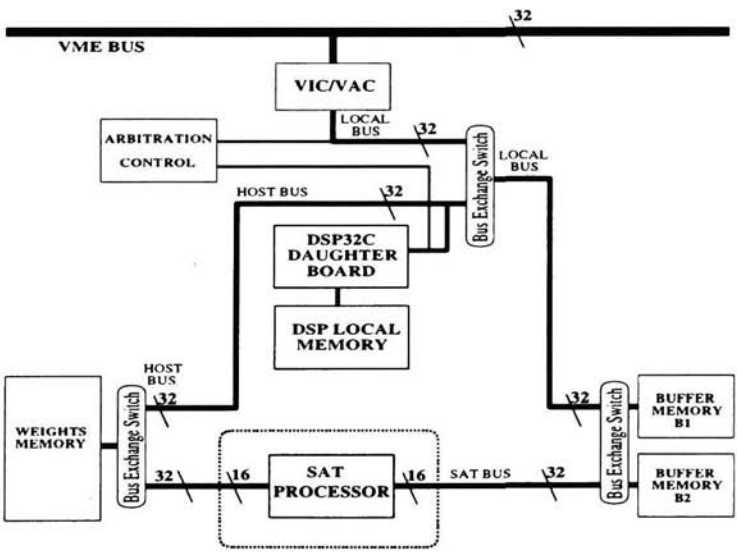


Figure 3: C-Node Architecture

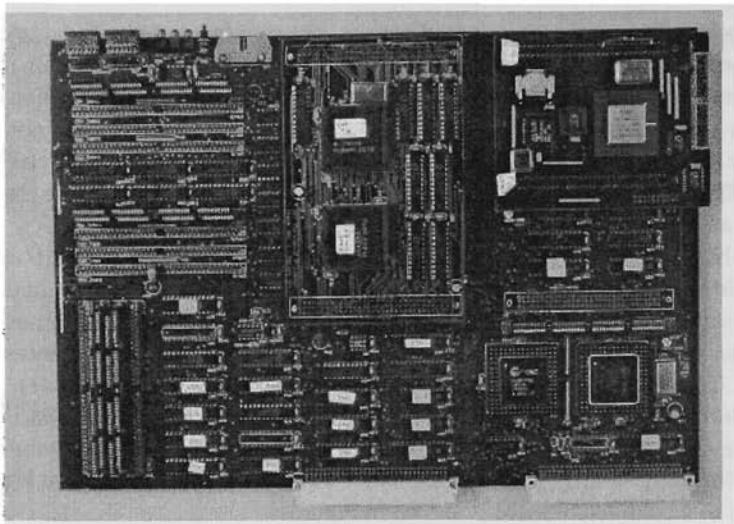


Figure 4: A C-Node Photograph

The C-node hosts a dedicated processor called the SAT processor (Sum And Threshold). The SAT recalls sixteen bits of active binary matrix in a single 175 nanosecond clock cycle. It is described fully in Section 4. The C-node also has three 25ns SRAM memories. The *DSP memory* is used solely for DSP code variable storage. The *buffer memories* are two memories that can be switched between buses. Memory one is accessible from either the DSP, or the S-node over the VME bus. Memory two is accessible from the SAT processor. The *weights memory* is a single memory that can be switched to either the DSP or VME bus, or to the SAT processor. The VME interface is provided through the VIC & VAC chipset.

The pre-processed data values applied to train and recall from the binary network are termed the index values, so termed because they index into the CMM. The index values for the binary pattern "0 1 0 0 1 0 0 0" are one and four as bits one and four are set (the first bit is classed as bit 0). The index values are generated on the S-node and then the DSP is informed that a transfer is required. The DSP configures the VIC & VAC as VME bus master which then perform a block transfer of the index values from the host computers memory into the buffer memory. During the training phase the DSP then uses these index values to train the CMM stored in the weights memory.

During a recall from the network the memories are switched so that the SAT processor can access the index values in the buffer memory, and can also access the weights memory. It is then interrupted by the DSP, after which the SAT will start re-

calling data from the matrix, placing results in the buffer memory. On completion it interrupts the DSP which swaps the buffer memories back. At this point the host computer will have generated new index values which are transferred to the second buffer memory. By this process there will always be a new set of index values in the buffer memory so that the SAT processor will continuously process data every time the buffer memories are swapped. The flow of data and control is shown in Figure 5, where the system consists of the S-node and three C-nodes.

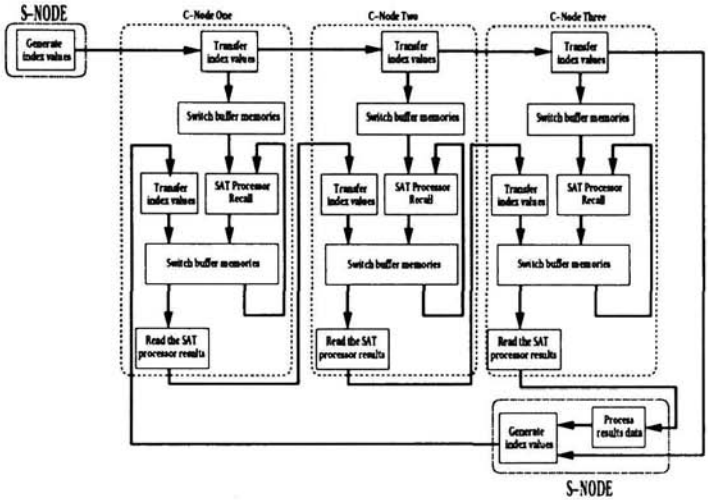


Figure 5: Data Flow on a Multiple C-Node System}

4 The Sum and Threshold Processor

The SAT processor has been designed to accumulate sixteen bits of the CMM in a single cycle. SAT performance is vital to the processing rate as the complexity of the recall using standard software techniques results in slow execution rates. The SAT has been designed so that it can process both single stage CMMs and two stage CMM networks, such as the ADAM network. It can also perform either Willshaw thresholding or *L-max* thresholding. It is currently implemented using two Actel FPGA devices.

The hardware of the SAT consists of four state machines, sixteen counters for summing the data from the weights memory, and other hardware for memory access and control. To recall data, the SAT is given an offset value within the control data which indicates the first address in the weights matrix. To this value the SAT adds the index value which results in the address to be accessed in memory. In Figure 6 the SAT is shown accessing the sixteen bits of matrix indicated by the first index value.

The sixteen bits it is accessing are then passed to individual counters which are clocked when the data is valid. This has the effect of incrementing those counters whose input is a "1". This entire cycle takes 175 nanoseconds. When all the active lines in a sixteen-bit row of the matrix have been summed the sixteen summed values are stored in the buffer memory and the process repeated on the next row of sixteen bits after zeroing the counters.

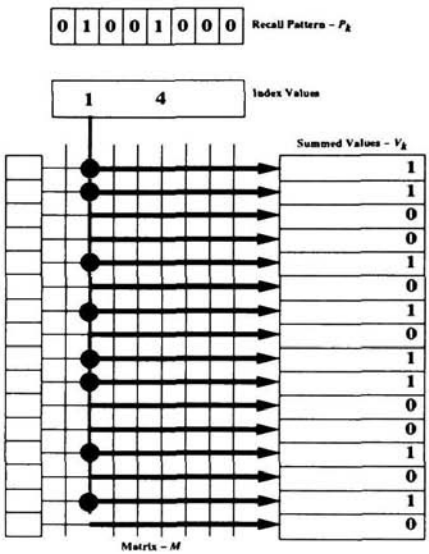


Figure 6: SAT Data Accumulation Method}

When all the summed values for all the rows have been written to the buffer memory, the SAT will either Willshaw or *L-max* threshold them. *L-max* thresholding begins by inspecting all the summed values to find the maximum value stored, this then becomes the current threshold value. All of the summed values are then re-checked. When the SAT processor finds values that equal L it saves the bit address of those values. If insufficient separator bits were found after the first thresholding iteration then the operation is repeated by finding the next highest summed value and using this as the new threshold value. This is repeated until all L separator bits have been recovered.

At the end of the thresholding operation, the results area of the buffer memory will contain a series of integers which identify which bits in the separator pattern were set. This is shown in Figure 7 where the values 0. 3. 9 etc. are stored in the buffer mem-

ory instead of the entire thresholded separator pattern. This is an improvement on version 1 of the SAT[8] which stored the entire bit pattern.

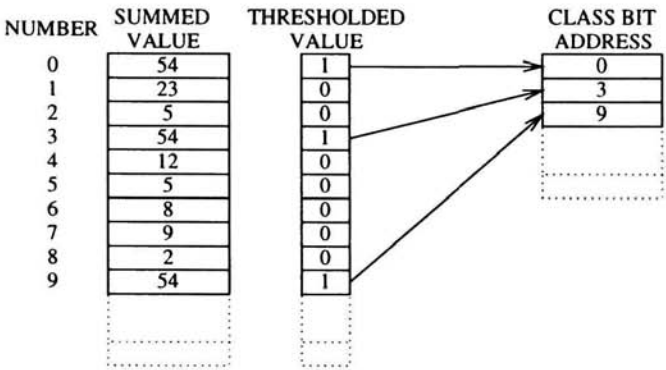


Figure 7: Thresholded Separator Storage}

5 Performance

5.1 Estimating Processing Duration

For a single CMM, the SAT processor has two equations to determine its execution speed which can be used to determine the speed of different applications. The equations were derived from the state machine diagrams. Equation 4 calculates S which in seconds is the SAT start-up time and summing of the data. Equation 5 calculates time T in seconds which is the time for thresholding the data. This equation should be omitted if global thresholding is done across multiple boards by the S-node. In the Equations, β is the number of CMM index values. For grey scale work this is the number of pixels per image, and for binary tupling it is $(pixels/n) \times 2^n$. The separator size is α. For thresholding, i is the number of iterations required to find all L bits of the separator pattern, ϕ is how often a summed value equals the stored threshold value per iteration.

$$S = 50 \times 10^{-9} \times \left[\frac{3.5\alpha\beta + 34\alpha}{16} + 27.5 \right] \quad (4)$$

$$T = 50 \times 10^{-9} \times [i(4.5\alpha + 3\phi)] \tag{5}$$

The above equations have been verified against the SAT processor running the specified 40 MHz clock.

By formula transformation the maximum number of index values, I, can be calculated for a given length of time τ and separator size. This is useful when calculating the quantity of data that can be processed at real-time rates. This is shown in Equation 6 for thresholding, and Equation 7 for the recall operation.

$$\delta = 50 \times 10^{-9} \times [i(4.5\alpha + 3\phi)] \tag{6}$$

$$I = \frac{16(\tau - 1.48125e^{-6} - \delta)}{50e^{-9} \times 3.5 \times \alpha} \tag{7}$$

In a typical scenario the separator pattern will be 64 bits with 6 bits set, all of which will be found on the first thresholding iteration. This data yields the information that 57140 index values can be processed in 40ms. If n-tupling of size four is used as a pre-processor, for example, then the number of input bits will be 228560, a 478 by 478 binary pixel image.

5.2 *Performance Comparison*

To appreciate the performance of the C-NNAP board it is necessary to compare its performance against that of a current (1996) range of high-performance computers. The first of these is a Silicon Graphics Indy workstation with a 133MHz R4600SC processor running Irix 6.2. The second is a Silicon Graphics Challenge server with one 150 MHz R4400SC processor also running Irix 6.2. The third is for the same server but with four processors, linear speed-up is assumed for all four processors, though this is optimistic. The results of the comparison are shown in the graph of Figure 8, where a weight is a single binary `1' or `0'.

In order to further emphasise the advantages of the C-NNAP architecture Table 1 shows the size of input image, post n-tupling, that can be processed at 25 Hz frame rates, for a single layer CMM network with an assumed input n-tuple size of 4 which produces one index value for four input bits, and a 64-bit separator pattern. The Table shows that a single C-Node can process nearly eight times the size of image store matrix as a high performance workstation

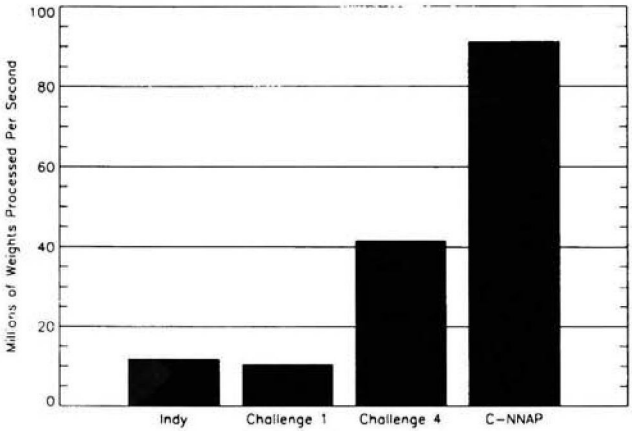


Figure 8: SAT Processor Performance.

Machine Type	Binary Pixels	Image Size
Indy	29500	172^2
Challenge, 1 Processor	26000	161^2
Challenge, 4 Processors	104000	323^2
C-NNAP, single C-Node	228000	478^2

Table 1: Frame Rate Processing Comparison

For larger problems than this it will be necessary to upgrade the relevant system. For example, doubling the processing rate of C-NNAP can be achieved by the addition of a further C-node, which currently costs in the region of \pounds 3000. If this value is cost C, it will take $2.7C$ to purchase a further indy to achieve double the processing rate. Upgrading the four processors of the Challenge to 200 MHz R10000 processors would currently cost $30C$. Therefore both of these options are significantly more expensive than a C-node upgrade.

6 Summary

This chapter has shown that binary neural networks are advantageous where either the processing rates are important, or where the input data is of high dimensionality. The chapter described in detail the C-NNAP architecture which is a general purpose processor of binary neural networks. It was shown that a single C-node C-NNAP can process data at nearly eight times the rate of a powerful workstation, at a fraction of the cost. It has therefore been shown that C-NNAP provides a powerful, yet cost effective platform for processing binary neural networks.

7 References

1. J. S. Alvus, A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *Transactions of the ASME*, pp. 220-227, Sept., 1975
2. I. Aleksander, T. J. Stonham, Guide to pattern recognition using random access memories, *Computers And Digital Techniques*, 2:29-40, 1979
3. J. Austin, T. J. Stonham, An Associative memory for use in image recognition and occlusion analysis. *Image an Vision Computing*, vol. 5, no, 4, pp. 251-261, Nov. 1987
4. J. Austin. Image pre-processing with a generalised grey scale n-tuple operator. Technical report YCS 102, The University of York, UK, 1988.
5. J. Austin. Distributed Associative Memories for high speed symbolic reasoning. *International Journal on Fuzzy Sets and Systems*, 1995.
6. W. W. Bledsoe, I. Browning. Pattern Recognition and Reading by Machine. *Proceedings of the Joint Computer Conference*, 1959.
7. R. Filer. Symbolic Reasoning in an Associative Neural Network. Master's thesis, University of York, UK, Sept. 1994
8. J. V. Kennedy, J. Austin. A Hardware Implementation of a Binary Neural Associative Memory. *4th International Conf. on Microelectronics for Neural Networks and Fuzzy Systems*, pp. 178-185, Torino, Italy, Sept. 1994.
9. D. Kustrin, J. Austin, A. Sanders. Application of Correlation Matrix Memories in a High Frequency Asset Allocation. *Fifth International Conf. on Artificial Neural Networks,* University of Cambridge, UK, July 1997
10. J. Kennedy. An Exploration into an Uncertain Reasoning Architecture, First year D. Phil Report, University of York, 1995.
11. D. Lomas. Improving Automated Postal Address Recognition. Master's thesis, University of York, UK, 1996.
12. M. Morciniec. The N-tuple Network. PhD thesis, The University of Aston in Birmingham, 1996.
13. M. Turner, J. Austin. Matching Performance of Binary Correlation Matrix Memories. *Neural Networks,* pp. 52-58, 1997.

14. M. Turner, J. Austin. A Neural Relaxation Technique for Chemical Graph Matching. *5th International Conf. on Artificial Neural Networks*, University of Cambridge, UK, July 1997.
15. D. J. Willshaw, O.P. Buneman, H.C. Longuet-Higgins. Non-holographic Associative Memory. *Nature* 222:960-962, June 1969.
16. *Proceedings of Weightless Neural Network Workshop 1995*, Canterbury, UK, 1995

This page is intentionally left blank

Author Affiliations

Dr. P J L Adeodato
Universidade Federal de Pernambuco - Brazil

Dr. I Aleksander
Imperial College of Science, Technology and Medicine - UK

Dr. J Austin
Mr. J Kennedy
Mr. K Lees
Dr. G Morgan
Dr. S E M O'Keefe
University of York - UK

Dr. J M Bishop
Mr. S K Box
Mr. J F Hawker
Dr. D A Keating
Dr. R J Mitchell
The University of Reading - UK

Dr. D L Bisset
Professor M C Fairhurst
Dr. G Howells
University of Kent at Canterbury - UK

Professor A C P L F De Carvalho
University of São Paulo at São Carlos - Brazil

Dr. S S Christensen
Dr. T M Jørgensen
Dr. C Liisberg
Risø National Laboratory - Denmark

Dr. T G Clarkson
Dr. Y Ding
King's College London - UK

Dr. L Hepplewhite
Dr. T J Stonham
Dr. P Ntourntoufis
Brunel University - UK

Dr. T Macek
Czech Technical University - Czech Republic

Dr. M Morciniec
Hewlett-Packard Laboratories - UK

Dr. R Rohwer
Prediction Company - USA

Dr. J G Taylor
King's College London - UK

RAM-Based Neural Networks

Edited by James Austin

This book presents an introduction, new methods and applications of RAM-based neural networks by leading researchers in the field. These networks are a class of methods for building pattern recognition systems characterized by the use of binary weights and one-shot learning. In addition, they have a direct mapping to digital hardware, which allows simple, fast implementation. The book provides a thorough grounding in the latest work, helping the reader to appreciate the state of the art in this technology.

ISBN 981-02-3253-5

3584 hc