

ASPECTS OF THE THEORY OF WEIGHTLESS ARTIFICIAL NEURAL NETWORKS

*A thesis submitted for the degree of
Doctor of Philosophy
and the
Diploma of Imperial College*

Panayotis Ntourntoufis

*Department of Electrical and Electronic Engineering
Imperial College of Science, Technology and Medicine
The University of London*

September 1994



ABSTRACT

This thesis brings together various analyses of Weightless Artificial Neural Networks (WANNs). The term *weightless* is used to distinguish such systems from those based on the more traditional *weighted* McCulloch and Pitts model. The generality of WANNs is argued: the Random Access Memory model (RAM) and its derivatives are shown to be very general forms of neural nodes.

Most of the previous studies on WANNs are based on simulation results and there is a lack of theoretical work concerning the properties of WANNs. One of the contributions of this thesis is an improvement in the understanding of the theoretical properties of WANNs.

The thesis deals first with feed-forward pyramidal WANNs. Results are obtained which augment what has been done by others in respect of the functionality, the storage capacity and the learning dynamics of such systems.

Next, unsupervised learning in WANNs is studied. The self-organisation properties of a Kohonen network with weightless nodes are examined. The *C-discriminator* node (CDN) is introduced and a training algorithm with spreading is derived. It is shown that a CDN network is able to form a topologically ordered map of the input data, where responses to similar patterns are clustered in certain regions of the output map.

Finally, weightless auto-associative memories are studied using a network called the General Neural Unit (GNU). The storage capacity and retrieval equations of the network are derived. The node model of a GNU is the Generalising Random Access Memory (GRAM). From this model is derived the concept of the *Dynamically* Generalising Random Access Memory (DGRAM). The DGRAM is able to store patterns and spread them, via a dynamical process involving interactions between each memory location and its neighbouring locations and/or external signals.

ACKNOWLEDGEMENTS

My thanks go first and foremost to my supervisor Professor Igor Aleksander for his help, encouragement and most especially his patience during the research and preparation of this Thesis.

I thank my colleagues from the Neural Systems Engineering Laboratory at Imperial, most especially, Dr. Eamon Fulcher and Dr. Catherine Myers, for their friendship and many discussions on important subjects, neural and other. Thanks go as well to the newer members of the group for their support during the write-up of this Thesis.

I thank everyone else who has given me support and advice, in particular, Dr. Feng Xiong and his family.

Last but not least, I thank my family for their love and continued support.

TABLE OF CONTENTS

ABSTRACT	2
ACKNOWLEDGEMENTS	3
TABLE OF CONTENTS	4
TABLE OF FIGURES	10
TABLE OF TABLES	12
TABLE OF PROOFS	13
LIST OF ABBREVIATIONS	14
 CHAPTER I. Introduction	 16
1.1. Systems studied in this Thesis	16
1.2. The origins of weightless neural computing	17
1.2.1. Introduction	17
1.2.2. Pattern recognition and classification techniques	17
1.2.3. Neural network modelling research	19
1.2.4. Study of Boolean networks	20
1.2.5. Automata Theory	21
1.2.6. Development of electronic learning circuits	21
1.3. Organisation of the Thesis	22
 CHAPTER II. Weightless artificial neural networks	 25
2.1. Introduction	25
2.2. Weighted-sum-and-threshold models	25
2.2.1. Node models	25
2.2.2. Training methods	26
2.3. Weightless neural nodes	28
2.3.1. The random access memory node	28
2.3.2. The single layer net node	29
2.3.3. The discriminator node	30
2.3.3.1. Definition	30
2.3.3.2. Prediction of the discriminator response	31
2.3.3.3. Internal representation of a pattern class	32
2.3.4. The probabilistic logic node	33
2.3.5. The pyramidal node	33
2.3.5.1. Definition	33
2.3.5.2. Training algorithms	34
2.3.5.3. Functional capacity	36

2.3.5.4. Generalisation performance	37
2.3.6. The continuously-valued discriminator node	38
2.3.7. The generalising random access memory node	38
2.3.7.1. The ideal artificial neuron	38
2.3.7.2. The GRAM model	38
2.3.7.3. Best matching and diffusion algorithm	39
2.3.8. The dynamically generalising random access memory node	40
2.3.9. Other weightless node models	40
2.4. Properties of weightless neural nodes	42
2.4.1. Introduction	42
2.4.2. Node loading	42
2.4.3. Generalisation by spreading	43
2.4.4. Generalisation by node decomposition	43
2.4.5. Introduction of a probabilistic element	44
2.5. Weightless neural networks	45
2.5.1. Network structure levels	45
2.5.2. Feed-forward weightless networks	46
2.5.2.1. Introduction	46
2.5.2.2. The single layer discriminator network	47
2.5.2.2.1. Description of the network	47
2.5.2.2.2. Learning and generalisation	47
2.5.2.2.3. Steck's stochastic model	49
2.5.2.3. The advanced distributed associative memory network	50
2.5.3. Recurrent weightless networks for associative memory	52
2.5.3.1. The sparsely-connected auto-associative PLN network	52
2.5.3.2. Fully-connected auto-associative weightless networks	55
2.5.3.2.1. Pure feed-back PLN networks	55
2.5.3.2.2. The GRAM perfect auto-associative network	57
2.5.3.3. The general neural unit network	58
2.5.4. Self-organising weightless neural networks and unsupervised learning	59
2.6. Summary	59
 CHAPTER III. The generality of the weightless approach	 62
3.1. Introduction	62
3.2. Generality with respect to the logical formalism	62
3.2.1. Neuronal activities of McCP and RAM neurons	62
3.2.2. RAM implementation of McCP networks	64
3.3. Generality with respect to the node function set	67
3.4. Generality with respect to pattern recognition methods	68

3.4.1. Introduction	68
3.4.2. The maximum likelihood decision rule	69
3.4.3. The maximum likelihood method	69
3.4.4. The maximum likelihood N -tuple method	71
3.4.5. The nearest neighbour N -tuple method	72
3.4.6. The discriminator network	73
3.5. Generality with respect to standard neural learning paradigms	74
3.6. Generality with respect to emergent property systems	75
3.7. Weightless versus weighted neural systems	75
3.7.1. Connectivity versus functionality	75
3.7.2. Ease of implementation	76
3.7.3. Learning and generalisation	76
3.7.4. Distributed and localised representations	77
3.8. Conclusions	77
 CHAPTER IV. Further properties of feed-forward pyramidal WANNs	 79
4.1. Introduction	79
4.2. Functionality of feed-forward pyramidal networks	79
4.2.1. Simple non-recursive formula	79
4.2.1.1. Introduction	79
4.2.1.2. Derivation	80
4.2.2. Approximations	83
4.3. Storage capacity	84
4.3.1. Definition	84
4.3.2. Methodology	85
4.3.3. The storage capacity of a regular pyramidal network	86
4.4. Dynamics of learning in pyramidal WANNs	89
4.4.1. Introduction	89
4.4.2. Previous work	89
4.4.3. The parity checking problem	90
4.4.4. Evolution of the internal state of the network during training	91
4.4.4.1. Transition probability distributions	91
4.4.4.2. Calculation of the transition probability distributions	92
4.4.4.3. Convergence of the learning process	93
4.5. Conclusions	96
 CHAPTER V. Unsupervised learning in weightless neural networks	 97
5.1. Introduction	97
5.2. Pyramidal nodes	97

5.3. Discriminator-based nodes	99
5.3.1. Introduction	99
5.3.2. The discriminator-based network	99
5.3.2.1. The network	99
5.3.2.2. The C-discriminator node	100
5.3.3. An unsupervised training algorithm	101
5.3.4. Explanation of the Equations (5.2) and (5.5)	103
5.3.5. Choice of a linear spreading function	103
5.4. Experimental results	106
5.4.1. Simulation 1	106
5.4.1.1. Introduction	106
5.4.1.2. The simulation	106
5.4.1.3. Temporal evolution of responses	109
5.4.1.4. Comparison with a standard Kohonen network	111
5.4.2. Simulation 2: uniform input pattern distribution	112
5.5. Comparisons with other weightless models	117
5.6. Conclusions	120
 CHAPTER VI. Weightless auto-associative memories	 121
6.1. Introduction	121
6.2. Probability of disruption and storage capacity	121
6.2.1. Assumptions	121
6.2.2. Probability of disruption of equally and maximally distant patterns	122
6.2.3. Experimental verification of Corollary 6.5	128
6.2.4. Equally and maximally distant patterns	129
6.2.5. Probability of disruption of uncorrelated patterns	130
6.3. Improving the immunity of the GNU network to contradictions	136
6.4. Conclusions	139
 CHAPTER VII. Retrieval Process in the GNU network	 141
7.1. Introduction	141
7.2. Relationships between pattern overlaps	141
7.2.1. Principle of inclusion and exclusion	141
7.2.2. Complementary overlaps	143
7.2.3. Useful corollaries	143
7.2.4. Higher order overlaps	144
7.3. Retrieval process in the GNU network	146
7.3.1. Definitions	146
7.3.1.1. Spreading function	146

7.3.1.2. Retrieval equations	147
7.3.2. Retrieval of two opposite patterns	148
7.3.3. General retrieval of three patterns	149
7.3.4. Example	154
7.4. Conclusions	156
 CHAPTER VIII. Dynamically generalising weightless neural memories	157
8.1. Introduction	157
8.2. From GRAM to DGRAM	157
8.3. The dynamically generalising random access memory	159
8.3.1. Notations	159
8.3.2. State transitions in a RAM	159
8.3.3. State transitions in a PLN	161
8.3.4. State transitions in a GRAM	163
8.3.5. State transitions in a DGRAM	164
8.3.5.1. Spreading levels	164
8.3.5.2. The states of a memory location	164
8.3.5.3. Write and remove operations	164
8.3.5.4. Interactions with neighbouring memory locations	165
8.3.5.5. General update equation	167
8.3.5.6. Stability of the dynamical learning process	167
8.4. Experimental results	168
8.4.1. The memory diagram	168
8.4.2. Example 1	169
8.4.3. Example 2: Transitory effect	171
8.5. Conclusions	171
 CHAPTER IX. Conclusions	173
9.1. Summary of the Thesis contributions	173
9.1.1. Mathematical analysis	173
9.1.2. Review of previous work	173
9.1.3. Feed-forward regular pyramidal WANNs	173
9.1.4. Unsupervised learning in WANNs	174
9.1.5. Storage capacity of the GNU network	175
9.1.6. Retrieval process in the GNU network	176
9.1.7. Dynamical spreading	177
9.3. Suggestions for future work	177
 REFERENCES	180

APPENDICES	190
Appendix A.	190
A.1. Proof of Lemma 4.1	190
A.2. Alternative proof of Theorem 4.1	191
A.3. Proof of Lemma 4.2	191
A.4. Proof of Theorem 4.2	192
Appendix B. Proof of Theorem 4.3	194
Appendix C. Calculation of the transition probability distributions (4.19)	198
Appendix D. Parameter variations in the C-discriminator network	207
D.1. Excitation radius	207
D.2. Noise level	210
Appendix E. Proof of Theorem 7.4	213
Appendix F. Published papers	218
F.1. Proceedings IJCNN-90, San Diego, June 1990	219
F.2. Proceedings IJCNN-91, Seattle, July 1991	225
F.3. Proceedings ICANN-92, Brighton, September 1992	231
F.4. Proceedings ICANN-93, Amsterdam, September 1993	235

Figure 5.15		117
Figure 5.16		119
Figure 6.1		123
Figure 6.2		130
Figure 7.1		150
Figure 7.2		151
Figure 7.3		154
Figure 7.4		154
Figure 7.5		155
Figure 8.1		158
Figure 8.2		158
Figure 8.3		161
Figure 8.4		162
Figure 8.5		163
Figure 8.6		165
Figure 8.7		166
Figure 8.8		168
Figure 8.9		170
Figure 8.10		171
Figure B.1		194
Figure B.2		194
Figure B.3		195
Figure D.1		207
Figure D.2		208
Figure D.3		209
Figure D.4		210
Figure D.5		211
Figure D.6		212

TABLE OF TABLES

Table 4.1		82
Table 4.2		87
Table 4.3		92
Table C.1		201
Table C.2		201
Table C.3		201
Table C.4		202
Table C.5		203
Table C.6		205
Table C.7		206

TABLE OF PROOFS

Lemma 4.1	82
Theorem 4.1	82
Lemma 4.2	83
Theorem 4.2	83
Theorem 4.3	90
Corollary 4.1	90
Theorem 6.1	124
Corollary 6.1	125
Corollary 6.2	125
Corollary 6.3	125
Corollary 6.4	216
Corollary 6.5	126
Corollary 6.6	127
Theorem 6.2	127
Theorem 6.3	131
Corollary 6.7	133
Corollary 6.8	134
Theorem 6.4	135
Theorem 6.5	137
Theorem 6.6	138
Theorem 7.1	142
Theorem 7.2	143
Corollary 7.1	143
Corollary 7.2	144
Theorem 7.3	144
Theorem 7.4	149

LIST OF ABBREVIATIONS

AAA	Al-Alawi algorithm
ADAM	Advanced Distributed Associative Memory
ANN	Artificial Neural Network
ART	Adaptive Resonance Theory
ASA	Aleksander Standard Algorithm
BBNM	Bledsoe and Browning N-tuple method
BMM	Best Match Machine
CDN	C-discriminator (Continuously-valued discriminator)
CMAC	Cerebellar Model Articulation Controller
C-RAM	Continuously-valued RAM
DGRAM	Dynamically Generalising Random Access Memory
DN	Discriminator node
DPBN	Derived Probabilistic Boolean Node
DPLM	Dynamically Programmable Logic Module
EBP	Error Back Propagation
FAE	Fused Adaptive Element
GDR	Generalised Delta Rule
GNU	General Neural Unit
GRAM	Generalising Random Access Memory
GSN	Goal Seeking Neuron
HLR	Hebbian Learning Rule
IAN	Ideal Artificial Neuron
KAM	Kanerva Associative Memory
LIBN	Linear Interpolating Boolean Node
LPNN	Lagrange Programming Neural Network
LU	Linear Unit
LWR	Local Weighted Regression
MBLC	Memory-Based Learning Controller
McCP	McCulloch and Pitts neuron
MDLR	Maximum Likelihood Decision Rule
MLM	Maximum Likelihood method
MLP	Multi-Layer Perceptron
MLNM	Maximum Likelihood N-tuple method
MPLN	Multi-valued Probabilistic Logic Node
NNHDM	Nearest Neighbour Hamming Distance Method
NNM	Nearest-Neighbour Methods

NNNM	Nearest Neighbour N-tuple Method
NSA	Neural State Automaton
NTA	Noise Test Algorithm
PCP	Parity Checking Problem
PIE	Principle of Inclusion and Exclusion
PLN	Probabilistic Logic Node
P-RAM	Probabilistic RAM
RAM	Random Access Memory
SDR	Standard Delta Rule
SLAM	Stored Logic Adaptable Microcircuit
SLLUP	Single-Layer Look-Up Perceptron
SLDN	Single Layer Discriminator Network
SLN	Single Layer Net
SR	Stimulus-Response
TOFM	Topologically-Ordered Feature Map
V-RAM	Virtual RAM
WANN	Weightless Artificial Neural Network
WST	Weighted-Sum-and-Threshold neuron

CHAPTER I. INTRODUCTION

1.1. Systems studied in this Thesis

An Artificial Neural Network (ANN) is an adaptive system made of a large number of simple processing elements, also referred to as nodes or neurons, which are interconnected according to some specific topology. Each element has an number of inputs, performs some simple function of these inputs and produces an output. A node receives its inputs either from the outputs of other nodes or from the external environment. Similarly, the output of a neuron can either be directly outputted to the environment or feed some other node inputs.

The *adaptation process* in an ANN usually takes the form of a training phase during which the nodes of the system are subjected to some *training algorithm* which modifies the function they perform. During this training procedure, the ANN learns to recognise stimuli from its environment, to classify them into different classes or to perform associations between these stimuli and appropriate responses, either in the form of outputs to the environment or as internal *state transitions*. During this *learning* process, the ANN also develops internal representations of its environment which are realised in the system by the creation of internal state structures.

This thesis is concerned with Weightless Artificial Neural Networks (WANNs). The basic processing element in a WANN is the Random Access Memory (RAM), which implements a truth table that records the responses of the node to different input signals. The adjective *weightless* is used to distinguish such systems from those based on the more traditional *weighted* McCulloch and Pitts model [McCulP43], in which the nodes of the system perform some linearly separable function by computing a weighted-sum-and-threshold of their inputs. Weightless nodes are also known as Boolean nodes, logic nodes, RAM-based nodes, cubic nodes [Gur89], look-up table nodes [Jud90], digital nodes [Ala90] or adaptable combinatorial elements [Ful93].

One of the thesis contributions is an improvement in the understanding of the theoretical properties of WANNs. The thesis investigates several WANNs analytically using mathematical tools. These are mainly probability theory, combinatorics, stochastic processes and automata theory. This philosophy of approach is in contrast to a whole branch of neural network research which bases its analyses on existing theories borrowed from Statistical Mechanics and Thermodynamics which involve the definition of variables (energy function, temperature, ...) not intimately related to the nature of the systems under

study. The other contributions of the thesis are the development of new node models, learning algorithms and spreading procedures for WANNs.

The next Section examines several fields of research which have influenced the development of connectionist models using WANNs. Then, Section 1.3 outlines the organisation of this thesis in its remaining chapters.

1.2. The origins of weightless neural computing

1.2.1. Introduction

The origins of weightless neural computing can be traced in developments from mainly 5 research areas: pattern recognition and classification techniques; neural network modelling research; the study of Boolean networks; automata theory; the development of RAM technology and electronic learning circuits. These research areas are succinctly examined in the next sub-sections.

1.2.2. Pattern recognition and classification techniques

Pattern recognition and classification techniques deal with problems typically involving the recognition or classification of unknown patterns into a certain number of classification classes, given a certain number of measurements on example patterns for the different classes. The example patterns are often referred to as *learning examples* or *training patterns*. Typically, in a problem involving z pattern classes $X_1, X_2, \dots, X_z$, m learning examples are available for each class. Some statistical estimation is then performed on the example patterns, which define a learning matrix M . When an unknown pattern x_p , representing class X_p , is presented for recognition, a vector $\bar{x}_p$ of measurements on x_p is formed. The recognition process aims at attaching a series of numerical scores for each pattern examined. This can then be expressed by the expression [Ste62]:

$$\bar{S}_p = (S_{p1} S_{p2} \dots S_{pz})^T = M \cdot \bar{x}_p, \quad (1.1)$$

in which S_{pj} represents the recognition score of x_p against the pattern class X_j , $\bar{S}_p$ is a score vector and r denotes the transpose operator. The set of j 's for which $\max_{1 \leq j \leq h} S_{pj}$ is attained, is denoted by J . If $p \in J$, the identification of x_p is said to be *correct*. Depending on whether J contains only one element or more than one element; the identification is said to be *correct without tie* or *with tie* [Ste62].

The problem of pattern recognition can also be approached by examining the relationships between the sets of patterns involved (e.g., [AleS79]), shown in Figure 1.1.

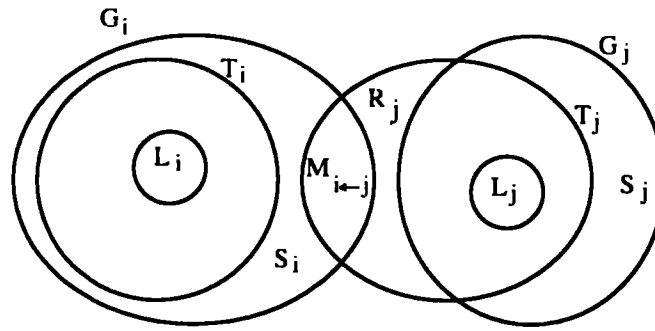


Fig. 1.1: Relationships between characteristic pattern sets in a pattern recognition problem.

The universal set of all patterns that could occur on a binary input matrix is denoted Ω . Then, for any recognition class X_i , L_i and G_i denote the training and generalisation sets, respectively. It is usually assumed that, for any class X_j , L_j is included in G_j and that the generalisation sets of 2 different classes have an empty intersection. The set of patterns to be classified and belonging to class X_j , or test set for class X_j , is denoted T_j . The set of patterns that will require classification is the union $\bigcup_{i=1}^C T_i$, whereas $\Omega \setminus \bigcup_{i=1}^C T_i$ is the set of meaningless spurious patterns.

Perfect recognition is achieved, for class X_j , if $T_j \subset G_j$. *Over-generalisation* occurs when $G_j \setminus T_j = S_j \neq \emptyset$. In that case, the spurious patterns in S_j , if they were used as test patterns, would be classified as belonging to class X_j . This is unimportant since these patterns do not belong to any of the test sets. More important is the possibility of *misclassification* which occurs if, for $i \neq j$, $G_i \subset T_j = M_{i \leftarrow j} \neq \emptyset$, in which case the patterns in $M_{i \leftarrow j}$ will be classified incorrectly as belonging to class X_j . Finally, if $T_i \cap \bigcup_{i=1}^r G_i = R_i \neq \emptyset$, the patterns in R_i will be treated as unclassifiable and will therefore be rejected. A *rejection* is usually considered as more favourable than a misclassification.

Most interesting for neural computing are *decision-making* methods which provide a measurable high reliability in decision making, as well as the possibility of low reliability components. Another important requirement is for such method is the recognition of patterns as a *whole*. This is referred to as *Gestalt recognition*. (e.g., [BleB59]) This is in contrast with methods involving the measurement of the *specific* characteristics of patterns, their analysis into parts, followed by a synthesis of the whole from the parts.

One such pattern recognition method, which was most influential for the development of weightless neural computing, was proposed by Bledsoe & Browning ([BleB59], [BleB62], [Ste62], [Ull73]). The method, which is referred to as Bledsoe and Browning N-tuple method (BBNM), was successfully applied to the recognition of hand-written characters; these constitute complex patterns with high individual variability,

The BBNM is highly general. It can be utilised to attenuate the information contained in highly-variable patterns, while at the same time retaining enough of the essence of the information to categorise the patterns [BleB59]. In other words, the resulting system exhibits both *generalisation* and *discrimination*.

The BBNM is reviewed in detail in Chapter 2, within the framework of its implementation in a WANN. In Chapter 3, as part of a discussion on the generality of the weightless approach, several other important pattern recognition methods, related to the BBNM, are compared. These other methods are the Maximum Likelihood method (MLM), the Maximum Likelihood N-tuple method (MLNM), the Nearest Neighbour N-tuple Method (NNNM) and the Nearest Neighbour Hamming Distance Method (NNHDM).

1.2.3. Neural network modelling research

McCulloch and Pitts' seminal paper [McCulP43] establishes important concepts which still today form the basis of artificial neural network research. The "all-or-none" character of nervous activity is pointed out and it is therefore suggested that neural events can be treated by means of *propositional logic*. For any logical expression satisfying certain conditions, a network can be found, which behaves in the fashion described by that expression. It is also shown that many particular choices among possible neuro-physiological assumptions are equivalent in terms of network behaviour.

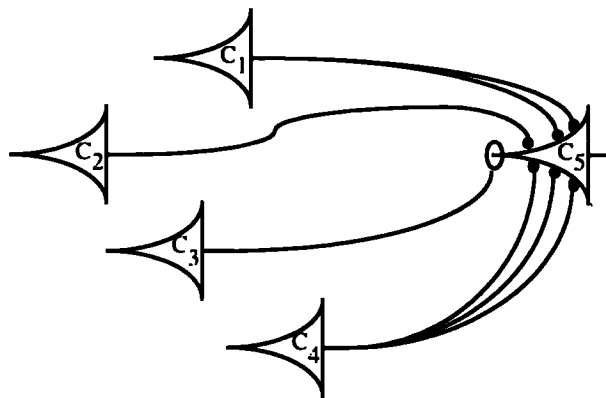


Fig. 1.2: A simple network of McCulloch & Pitts formal neurons. Neurons c_1 , c_2 and c_4 have excitatory synapses on neuron c_5 and neuron c_3 is an inhibitory neuron for c_5 .

The neuron model proposed is very simple. In the following, it is referred to as McCP neuron. Each neuron is normally quiescent, outputting 0, until a certain minimum number of its inputs are activated, in which case it fires, outputting 1. Inhibitory inputs exist, whose activity can prevent a neuron from firing. Weights are defined, implicitly, as the number of connections that a neuron receives from another neuron. The behaviour of any

network of such neurons can be described as a inclusive disjunction of logical minterms, and any such proposition can be computed by some network. Figure 1.2 shows a simple network of McCP neurons.

In the weightless approach, the McCulloch and Pitts logical formulation is extended such that the behaviour of the neurons can be described as a inclusive disjunction of *all* possible minterms formed by the logical values of their inputs.

In [McCulP43], no mention is made of what changes should be made to the nodes for learning to take place. In the weightless approach, learning is viewed as resulting from the changes to the logical propositions executed by the neurons. A straightforward consequence of this interpretation of learning is to associate each node with a look-up table defining a logical proposition.

Nervous activity laws can be formulated both for McCP neurons and weightless neurons. This is carried out in Chapter 3, in which two expressions are derived for the neuronal activities of McCP and RAM neurons. Usually, the two types of neuronal models are compared by expressing the activation laws in an analytical form (e.g., [Gur89], [Mar89] or [GorT88]). In Section 3.3, the activities are expressed using a *logical formalism*. For the McCP, it is the formalism that was originally used by McCulloch and Pitts. For the RAM, it is the most natural formalism.

1.2.4. Study of Boolean networks

An important body of research which has influenced weightless neural computing is the study of random Boolean networks (e.g., [Kau69], [FogGW82] or [Ale73]). Usually, each node performs a Boolean function chosen randomly amongst all possible functions of its inputs. Alternatively, the analysis can be performed on RAM networks, in which the RAM contents is arbitrary. These networks are referred to as *natural networks* [Ale87].

Usually, there is no training involved and the objective is to define the most likely state structure of such a system [Ale83a]. Important results can be obtained, such as the fact that random feed-back networks with fixed function and topology tend to enter limit cycles.

In [Ale83a], it is shown that in a network with fixed input, or autonomous automaton, the set of all states is partitioned into confluents (connected states). Each confluent consists of precisely one cycle and tree-like structures leading to it. The most likely number of states, in an autonomous network, that lead into some given state can be calculated. It is also possible to estimate the most likely distribution of transient and cyclic activity and to show that, when the network inputs are allowed to vary, such activity is most sensitive to changes in input for most parameter values. The fact that the state structure of a random Boolean network is very sensitive to its inputs indicates that there is a freedom within the system to learn desired responses without need to impose a particular network connection configuration. This ability of Boolean networks to learn is important and this property is

exploited in WANNs to enable the system to develop useful computational state structures,

1.2.5. Automata Theory

As already pointed out in the previous sub-section, the study of Boolean networks with *feed-back* leads naturally to the study of their emergent state structures. The system is then seen in the light of automata theory (e.g., [AleH76], [HopU79] or [CarL89]). When the network topology and the functions performed by the nodes of the network are fixed, the system has the structure of a finite state automaton. If the node connectivity or the node functions have a probabilistic character, then the system becomes a probabilistic automaton.

1.2.6. Development of electronic learning circuits

The weightless paradigm also finds its origins in the development of economical electronic components used in learning circuits. In the early days of neural computing research, electrical circuits implementing weighted neuron models required the use of bulky, expensive, and slow components such as Rosenblatt's motor-driven potentiometers or Widrow's memistors [Ale70]. As an alternative, electronic circuits capable of learning and performing logic functions were proposed by Aleksander as models of neuron elements. Some of these components are reviewed.

In [Ale65], a transistor circuit called Fused Adaptive Element (FAE) is described and demonstrated in adaptive logic experiments. An FAE operates in 2 phases. During a *teach* phase, the device is exposed to the complete set of inputs and corresponding outputs of a required logic function. The inputs are first decoded into minterms lines by some external decoder circuit and the FAE selects and memorises the required minterms. The minterm selection is implemented by burning low-current fuselinks corresponding to the minterms of the required logic function¹. During an *operate* phase, the circuit provides an output which is the required logic function of the inputs.

In [Ale70], the development of large-scale integrated microcircuits is reported, of which the Stored-Logic Adaptable Microcircuit (SLAM) is the building block. A SLAM has essentially the functionality of a FAE augmented with the input decoder circuit. This is shown in Figure 1.3. The SLAM has variable-function properties similar to those of neuron models of the McCulloch & Pitts variety. Also, it is a learning circuit:

"A fascinating aspect of this device is that, even though it is basically a simple storage circuit, in use, it becomes a variable-function logic element, the function of which may be adjusted during training" [Ale70].

¹In [Ale65], FAE circuits were envisaged, which allowed for a gradual change in the resistance of the links. It was also suggested that these circuits could be used in continuously self-organising binary systems. A similar kind of node, called C-RAM, will be introduced in Chapter 5, as a contribution to this thesis.

However, this device does not generalise. It is the behaviour of networks of such devices that exhibit generalisation properties.

Based on the SLAM developments, several machines were constructed, the aim being to obtain a learning machine large enough and flexible enough to be used as a generator of useful, special-purpose processors [Ale70]. The SOPHIA machine (1969) consisted of a 36-input circuit of 12 SLAM-8s, a SLAM- 2^N referring to a device with N input terminals. The MINERVA machine (1972) consisted of a circuit of 1024 SLAM-16s.

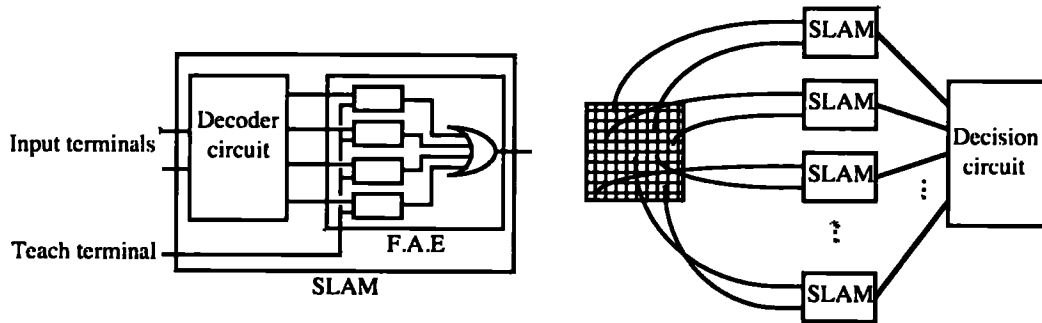


Fig. 1.3: FAE, SLAM and network of SLAMs.

As Random Access Memories (RAM) became more widely available as standard storage devices, their use was adopted as the building block of subsequent weightless neural machines. The WISARD machine (1981) is such a machine, with 32,000 RAM-8s [AleTB84]. WISARD is a general purpose parallel processing machine which has its origin in the interpretation of the basic addressing computation in the BBNM scheme as analogous to that taking place in a Random Access Memory when it is addressed by some binary input pattern. The machine has been used for high-performance image identification [Ale83b]. In [AleW85], the WISARD architecture is used in the context of adaptive window processing. The WISARD has also been implemented as a commercial machine [ComRS86], which can be trained on or classify 512×512 pixel binarised television images at a rate of 25 per second.

1.3. Organisation of the Thesis

The remainder of this thesis is organised in eight chapters. These are outlined.

In Chapter 2, the relevant literature is reviewed and the necessary background on WANNs is presented. The chapter reviews the major models of weightless nodes and their properties. This is followed by a review of the most important weightless network architectures. Work on McCp neuron models is also reviewed but in the context of the weightless models examined. For some weightless and weighted models, more specific

review is reported to subsequent chapters of the thesis.

In chapter 3, the generality of the weightless approach is argued. The generality of the logical formalism is first demonstrated. The choice of the node function set is then discussed. This is followed by a comparison of different pattern recognition methods related to the operation of a single-layer discriminator network. Weightless implementations of standard learning paradigms are briefly reviewed followed by a discussion on emergent property systems. Finally, several points of comparison between weighted and weightless neural systems are made.

Chapter 4 deals with feed-forward WANNs. The main structure studied is the pyramidal feed-forward network. Given first are some further results on the functional capacity of these networks (non-recursive formulae, approximations). Then the storage capacity of these networks, or the number of associations that they can store, is investigated for very simple cases. Finally, the dynamics of learning in these networks is studied and the state transition probabilities during training are calculated. The proofs of a series of results are given in appendices A, B and C.

Chapter 5 is concerned with unsupervised learning in WANNs. The self-organisation properties of a Kohonen network are studied, in which the nodes are weightless. The *C-discriminator* node is introduced. It is shown that a C-discriminator network is able to form a topologically ordered map of input data, where responses to similar patterns are *clustered* in certain regions of the output map. It is possible to predict the network responses to different input patterns considering the overlap areas between patterns. The algorithm used to train the C-discriminator network uses a *spreading algorithm* which affects memory locations not addressed by the input training patterns. The aim of the spreading operation is to diffuse the information acquired by the node from the training patterns in order to be able to generalise its knowledge to "unseen" patterns.

Chapter 6 of the thesis deals with weightless auto-associative memories. A network called the General Neural Unit (GNU) [Ale90a] is studied for its ability to perform auto-association. The nodes of a GNU are Generalising Random Access Memories (GRAMs). A GRAM operates in three phases. The *training* phase is the operation by which the GRAM records addresses with their required response. This is similar to the *write* mode of a RAM. The training phase is followed by a *spreading* phase. Finally, during the *use* phase, the node makes use of its stored information to produce an output. The disruption probabilities between training patterns in the GNU are calculated and the storage capacity of the network is derived. A method for improving the immunity of the GNU network to disruptions is also proposed, which uses a network based on the GNU but with discriminator nodes instead of GRAMs.

Chapter 7 studies the retrieval process in the GNU network. First, the relationships between pattern overlaps are derived and several useful corollaries are formulated, using the principle of inclusion and exclusion. Then, the retrieval equations of the GNU network

are derived.

Chapter 8 introduces the concept of the *Dynamically Generalising* Random Access Memory (DGRAM). The DGRAM is derived from the GRAM, in which the *training* and *spreading* operations are performed in a single phase. The DGRAM is able to store patterns and spread them, via a dynamical process involving interactions between each memory location and its neighbours and/or external signals. The DGRAM possesses certain advantages over the GRAM. First, it is possible to store additional patterns even after the spreading of previous patterns. Second, it is possible to distinguish between trained and spread patterns. And finally it is possible to remove a trained pattern and its associated spread patterns without disturbing the rest of the stored patterns.

Finally, Chapter 9 is devoted to a summary of the Thesis contributions and to suggestions for future work.

CHAPTER II. WEIGHTLESS ARTIFICIAL NEURAL NETWORKS

2.1. Introduction

This review chapter begins with a brief Section on the weighted-sum-and-threshold neuron model. The Hebbian learning paradigm and derived learning methods are also briefly examined. Then, Section 2.3 presents a review of WANN node models, detailing their memory structure, learning and generalisation properties. Section 2.4 recapitulates on the major properties found in weightless nodes. This is followed, in Section 2.5, by a review of the different types of WANN structures and a review of major studies of network behaviours. Throughout the chapter, work on McCp neuron models is also reviewed but in the context of the weightless models examined.

2.2. Weighted-sum-and-threshold models

2.2.1. Node models

Most artificial neural network learning paradigms centre around the weighted-sum-and-threshold neuron model. A Weighted-Sum-and-Threshold neuron (WST) is a generalisation of a McCulloch and Pitts' neuron. Such a neuron or node j has n inputs $x_1, x_2, \dots, x_i, \dots, x_n$. A real-valued weight w_{ji} is associated with each x_i . Typically, the weight values are restricted to a predetermined range, for instance $[0,1]$ or $[-1,1]$ [Mye90]. Only the case $[0,1]$ is considered here. An *activation* level a_j is calculated from the weighted sum of the inputs,

$$a_j = \sum_{i=1}^n x_i w_{ji} \quad (2.1)$$

and the output y_j takes one of the binary values, 1 or 0, depending on whether the activation a_j is above or below some real-valued threshold θ_j :

$$y_j = (a_j > \theta_j). \quad (2.2)$$

In (2.1) and (2.2), the variables x_i and y_i are assumed to take binary values 1 or 0. The model generalises easily to accommodate the use of analog inputs and outputs. Having

analog inputs and outputs also allows for the development of gradient descent learning algorithms (see later in this Section) and the use of mathematical analysis to study the behaviour of networks of WST neurons. A general expression for the output function of the neuron would then be:

$$y_j = f\left(\sum_{i=1}^n x_i w_{ji} - \theta_j\right), \quad (2.3)$$

in which x_i and y_j take analog values within some range and f is an *output function*.

The function f is chosen to be monotonically increasing with its argument. Often, f is chosen to be sigmoidal. The choice of a sigmoid function enables efficient training and analysis while preserving the desired “all-or-none” character of the binary formulation.

When f is the identity function, (2.3) becomes

$$y_j = \sum_{i=1}^n x_i w_{ji} - \theta_j. \quad (2.4)$$

Neurons with output function (2.4) are often referred to as Linear Units (LU). When θ_j is set to 0, (2.4) represents the function performed by the neurons of a standard Kohonen network [Koh89a]. More about this case will be said in Chapter 5, when the C-discriminator model is introduced as a contribution to this thesis. Also, the mathematical developments of Chapter 3 will show how a formally similar expression to (2.4) is derived in the maximum likelihood pattern classification method, between the conditional probability of class membership and the estimates of pixel distribution in the input patterns.

2.2.2. Training methods

This sub-section examines some of the standard methods used for training WST nodes to perform desired input-output associations or mappings.

Hebbian learning [Heb49] is a form of learning derived from the simplification of observed biological behaviour which encourages the co-activity between nodes. In a ‘weighted’ implementation, a Hebbian Learning Rule (HLR) can be defined, which strengthens weights between connected nodes that are frequently co-active. This can be expressed as

$$\Delta w_{ji} \propto x_j \cdot x_i,$$

in which Δw_{ji} denotes the variation of the weight from node i to node j , and x_j and x_i denote the output values of nodes j and i , respectively.

The Standard Delta Rule (SDR) [WidH60] is derived from the HLR. The SDR is more useful, in practice, than the HLR, as it expresses the weight variation as being proportional to an error signal δ_j which is the difference between the node's desired output d_j and actual output x_j :

$$\Delta w_{ji} = \eta \cdot (d_j - x_j) \cdot x_i = \eta \cdot \delta_j \cdot x_i,$$

in which the coefficient η can be interpreted as a *learning rate*.

The Generalised Delta Rule (GDR), or Error Back Propagation (EBP) algorithm [RumHW86], allows the training of multi-layer feed-forward networks of WST nodes using a gradient descent scheme. The rule requires the nodes to perform some non-decreasing differentiable output function f_j of the form

$$x_j = f_j(a_j),$$

in which a_j is given by (2.1). The GDR takes exactly the same form as the SDR:

$$\Delta w_{ji} = \eta \cdot \delta_j \cdot x_i,$$

The determination of the error signals δ_j is an recursive process which starts with the output nodes of the network:

$$\delta_k = (d_k - x_k) \cdot f'_k(a_k),$$

using the notation

$$f'_k(a_k) = \frac{\partial x_k}{\partial a_k},$$

and back-propagates to the hidden nodes of the network:

$$\delta_i = f'_i(a_i) \cdot \sum_k \delta_k \cdot w_{ki}.$$

Other methods exist for training WST or LU nodes (e.g., [RumM86], [Koh89a]) and a host of learning paradigms have been implemented using these node models and their associated training algorithms. Amongst the most important of these paradigms are the multi-layer perceptron [RumM86], Kohonen's topological feature map [Koh82b], Hopfield's network [Hop82], the Boltzmann machine [AckHS85], Grossberg adaptive

resonance theory [CarG89], Barto's reinforcement learning [BarSA83], Jordan's recurrent network [Jor88] and Kanerva's Sparse Distributed Memory [Kan88]. In Section 3.5, as part of the discussion on the generality of the weightless approach, references are given for the implementation of these standard learning paradigms using weightless nodes instead of LUs or WST neurons.

2.3. Weightless neural nodes

2.3.1. The RAM node

The Random Access Memory (RAM) is the basic component of WANNs. A weightless RAM has the structure of a bit-organised random access memory, in which the N address lines correspond to synaptic inputs of the neuron and the data out terminal is a modelisation of an axonic output. The data in terminal is sometimes referred to as 'dominant synapse' [Ale83b]. Figure 2.1 shows a schematic representation of a RAM node.

Equivalently, a RAM node can be viewed as implementing a look-up table, the entries of which are determined by the RAM inputs. A pattern, presented to the RAM inputs, addresses some location in the table, and the output of the node is dependent on a value stored at the addressed location. Also the RAM can be viewed as implementing a universal logic element, providing all the minterms of its input terminals [Ale65].

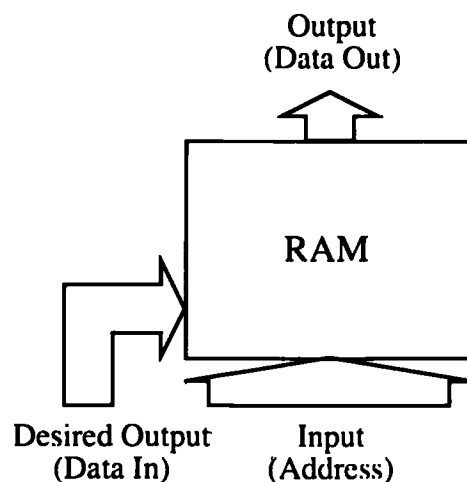


Fig. 2.1. Schematic representation of a RAM node.

Learning in a RAM node is done during a *training* phase. It simply consists of storing the desired output value for each possible input pattern at the associated address. It is therefore trivial to memorise associative data in a RAM. During a *use* phase, an input pattern addresses a memory location in the RAM and the value stored at that location is outputted.

A RAM does not generalise; an appropriate response must be stored for every possible input. The node can learn to execute all 2^{2^k} binary functions of its inputs, whereas the WST is restricted to the set of linearly separable functions.

The inputs to a RAM node are binary variables. It is possible to extend the model to accommodate inputs with 'grey-level' values. In order to do so, the grey-level inputs need to be coded in a suitable way. Binary and Gray codes are unsuitable because they distort the generalisation characteristics of the network with respect to properties of the data in the physical world [AleS79]. The thermometer coding gives good results but a large input space is required. A simple solution is to apply a threshold to the grey-level inputs. In [Aus88], a rank coding method is applied to RAM networks trained to classify different textures. More recently, the use of the CMAC coding scheme (CMAC is the acronym for 'Cerebellar Model Articulation Controller' [Alb75a] [Alb75b]) has been investigated in [AllK93].

2.3.2. The Single Layer Net node

A Single Layer Net (SLN), as defined in [AleS79], consists of a *decomposed structure* or array of K RAM nodes. Each of those RAMs is addressed by a N -tuple which is a number N of bits taken from an array of m binary values forming the input to the SLN. Thus, it holds that

$$m = K \cdot N.$$

The RAMs form a single layer network and their outputs are used as inputs to some *output function*. Figure 2.2 shows a schematic representation of a SLN with the logical output function AND.

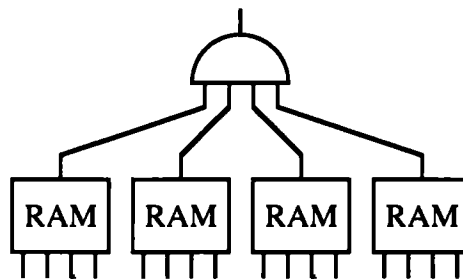


Fig. 2.2. A SLN node with parameters $N = K = 4$ and with output function AND.

The adaptive parameters of an SLN are the values stored in the memory locations of its RAMs. They are updated through a training procedure. Before training begins, all memory locations are set to 0. The training of a pattern in the SLN consists of storing a 1 at the

location addressed by the pattern in each RAM of the SLN.

The SLN exhibits generalisation properties. In [AleS79], the generalisation ability of an SLN node is discussed in terms of the size of the generalisation set G_i , as defined in Section 1.2.2. The size of G_i is affected by the choice of output function, the choice of input mapping and the diversity of the patterns in the training set. The output function strongly affects the size of the generalisation set. For instance, if the logical AND is used as output function, the generalisation set G_i of class X_i is made of all combinations of each RAM's addressing sub-patterns. The size of G_i is given by:

$$\|G_i^{AND}\| = \prod_{j=1}^K k_j,$$

with k_j the number of patterns seen by the j -th RAM during training. If the output function OR is used, the size of G_i becomes

$$\|G_i^{OR}\| = 2^{(K-1) \cdot N} \sum_{j=1}^K k_j$$

and it can easily be seen that

$$\|G_i^{OR}\| \gg \|G_i^{AND}\|.$$

The input mapping affects the way features are picked up. For instance, the connection of RAMs to common features in the patterns of the training set reduces the size of G_i . However the input mapping is usually chosen to be random and such that each RAM input 'sees' one different pixel of the overall input pattern. The diversity of the patterns in the training set affects the size of the generalisation set. Indeed, an increase in the size of the training set results in an increase in the size of G_i .

2.3.3. The discriminator node

2.3.3.1. Definition

A discriminator node (DN) is defined as being an SLN node with parameters K , N and with output function consisting of the algebraic sum of the values addressed in the RAMs of the node. Figure 2.3 shows a schematic representation of a discriminator with parameters $K = N = 4$.

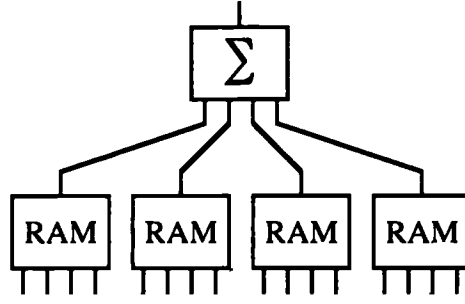


Fig. 2.3: A discriminator node ($N = 4$ and $K = 4$).

The node output, or response r_* , takes 'analog' values in the range $[0, 1]$ and is obtained by calculating the proportion of RAMs in the discriminator, which output a value 1 in response to an input pattern. Thus,

$$r = \frac{1}{K} \sum_{k=1}^K f_k \quad (2.5)$$

in which f_k denotes the binary value addressed in RAM k of the discriminator. The response of a discriminator is often quoted as a percentage. It is, in some way, a measure of similarity of an unknown pattern to each of the patterns in the training set [AleM90].

The discriminator node has the ability to generalise. Patterns similar but not identical to those in the training set generate responses similar to those generated by the patterns in the training set.

2.3.3.2. Prediction of the discriminator response

The discriminator response can be predicted [Ale83b] provided that all overlaps between all trained patterns and test patterns are known. For one stored pattern T_1 , the response $r(U)$ of the discriminator to an unknown pattern U can be approximated as

$$r(U) \approx \omega_1^N. \quad (2.6)$$

in which ω_1 represents the overlap between T_1 and U . When t patterns $T_1, T_2, \dots, T_t$ have been stored, the response to pattern U can be approximated as [AleM90]:

$$\begin{aligned} r(U) \approx & \omega_1^N + \omega_2^N + \dots + \omega_t^N \\ & - \omega_{12}^N - \omega_{13}^N - \dots - \omega_{1t}^N - \omega_{23}^N - \omega_{24}^N - \dots - \omega_{2t}^N - \dots - \omega_{(t-1)t}^N \\ & + \omega_{123}^N + \omega_{124}^N + \dots + \omega_{12t}^N + \omega_{134}^N + \dots + \omega_{(t-2)(t-1)t}^N \\ & \vdots \\ & \vdots \end{aligned}$$

$$+ (-1)^{i-1} \omega_{12 \dots i}^N, \quad (2.7)$$

in which $\omega_{1,2 \dots i}$ represents the overlap between U , T_{i_1} , T_{i_2} , ..., and T_{i_i} . Equation (2.7) can be approximated by (2.6) if U is closer, in Hamming distance, to T_1 than to any other training pattern. In [Ale84b], a very simple analysis of the discriminator's performance is made, geared towards the establishment of design choices for engineers and it is argued that a single discriminator has a discriminatory power that increases sharply and monotonically with N . In [AllJ92], further analysis of the inherent distance metric in the discriminator memory space is performed.

2.3.3.3. Internal representation of a pattern class

In the following development, a pattern x is seen as a ordered set of m elements that can take the values 0 or 1. The m elements of x are divided, randomly, into K randomly chosen and mutually exclusive ordered sets of N elements each. Therefore x can be seen as a set of K N -digit binary numbers requiring $K \cdot 2^N$ memory addresses, where $N \cdot K = m$. This leads to the definition of a column vector $\bar{x}$ of $K \cdot 2^N$ elements which has K elements equal to 1 and the rest of the elements equal to zero:

$$\bar{x} = \left(a_1 a_2 \dots a_{2^N} a_{2^N+1} a_{2^N+2} \dots a_{2 \cdot 2^N} \dots a_{(K-1)2^N+1} a_{(K-1)2^N+2} \dots a_{K \cdot 2^N} \right)^T, \quad (2.8)$$

with

$$a_{(r-1)2^N+i+1} = \begin{cases} 1 & \text{if the } i\text{th location is addressed in RAM } r \\ 0 & \text{otherwise} \end{cases}$$

$\bar{x}$ is the internal representation of pattern x inside the discriminator memory space. In [AllJ92], each N -tuple is interpreted as forming a sub-space of the input pattern space. Using the definition (2.8), the learning of a class X of t patterns results in the definition of a vector $\bar{X}$ such that

$$\bar{X} = \sum_{k=1}^t \bar{x}_k,$$

in which $\bar{x}_k$ denotes the k -th representation of X and $\sum$ is used to denote the logical addition of the corresponding elements in vectors $\bar{x}_k$. $\bar{X}$ is the internal representation of pattern class X inside the discriminator memory space.

2.3.4. The Probabilistic Logic node

In contrast with the single-bit RAM model, the input of a Probabilistic Logic Node (PLN) addresses a b -bit word and outputs a b -bit binary number which can be thought of as representing some real number B in the interval $[0,1]$. Alternatively, B can be interpreted as the *firing probability* at a binary node output, that is, the probability that the node outputs 1. This digital interpretation enables the ‘closure’ between the output of one node and the input to another [Ale88].

The PLN is trained with a local training rule based on a reward-punishment scheme. Reward consists of incrementing the value of B towards 1 if it is greater than 0.5 or towards 0 if it is smaller than 0.5. Punishment means incrementing or decrementing B always towards 0.5.

A particular version of the PLN is when B only takes 3 values: 0, 0.5 and 1. In that case, the value 0.5 is often denoted u or ‘unknown’, to indicate that, when this value is addressed, the node will fire or not fire (i.e. respond 1 or 0) with equal probability. Before any training takes place in the PLN, all memory locations are usually set to the u value. The terminology PLN is often reserved for that 3-level version of the probabilistic logic node (e.g., [Kan86] or [KanA89]). Versions with more than 3 values for B are referred to as MPLNs (multi-valued probabilistic nodes) (e.g., [Mye88b], [Mye89] or [Mye90]).

PLN networks have been found to learn faster than equivalent WST networks, at least for networks requiring a small number of nodes [Mye90]. Furthermore, PLN networks reach peak performance within a small number of passes through the complete training set [BisFF89].

2.3.5. The pyramidal node

2.3.5.1. Definition

A pyramidal node, or pyramid, is defined, in its canonical form, as a network of RAMs or PLNs arranged in a hierarchical non-overlapping tree structure. In such a structure, no external input is viewed by more than one input node. The two independent structural parameters of the pyramid are the number of inputs per RAM, N , and the number of layers in the structure, or depth, D . The total number of inputs of a pyramid is N^D and the number $M_{N,D}$ of RAMs is given by the expression

$$M_{N,D} = \frac{N^D - 1}{N - 1} \quad (2.9)$$

Such a node is represented in Figure 2.4, in the case $N = 4$ and $D = 2$.

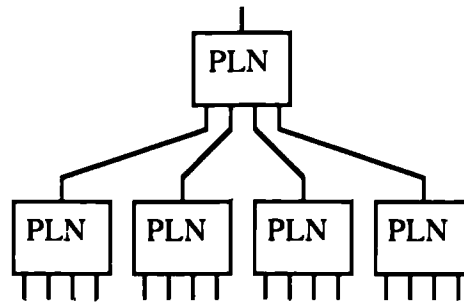


Fig. 2.4: A pyramidal weightless node ($N = 4$ and $D = 2$).

From the above description, it can be seen that the pyramid is a decomposed structure and therefore its functionality is constrained. The pyramid cannot store a specific response to each input pattern, in a way that a RAM would, and it must therefore group similar patterns together in the same response class. The parameters N and D are two powerful factors in determining generalisation and its complement, discrimination. An increase in the value of N yields an increase in discrimination and therefore a decrease in generalisation. Correspondingly, a decrease in N has the opposite effect: it increases generalisation and therefore decreases discrimination. The depth D of the pyramid, together with N , affect its *functional capacity* or number of distinct Boolean functions that the pyramid can perform in response to training. This is further examined in Section 2.3.5.3.

Experimental results have shown that pyramidal nodes suffer from a poor quality of generalisation, that is, input patterns close in Hamming distance to stored patterns, do not always produce a similar response to the stored patterns. For example, in [BisFF89], experimental evidence shows that deep pyramids (large D) with low node connectivity (small N) perform very poorly on a problem of machine-printed character recognition. The generalisation performance of pyramids is further discussed in Section 2.3.5.4.

Apart from affecting the functionality and generalisation properties of the pyramid, the node decomposition implies also that, for the hidden nodes of the pyramid, the desired output is not explicitly provided by the training set. This is one of the reasons for the replacement of the RAMs in the pyramid by PLNs. The ability of PLNs to 'guess' the value to output at a node is an essential feature of the pyramid training algorithms. Several related algorithms are reviewed in the following sub-section.

2.3.5.2. Training algorithms

Several algorithms have been proposed in order to train pyramidal nodes. They all centre around an overall reward/punishment scheme: all the nodes in the pyramid are rewarded

whenever the output node outputs the correct value and punished whenever it fails to do so consistently [Ale88]. Kan [KanA89] proposes the simple rule that consists of ‘freezing’ the current values of the PLN outputs as soon as the pyramid provides the correct output, in such a way that these output values are firmly associated with their current inputs.

The PLN pyramid standard training algorithm, referred to as Aleksander Standard Algorithm (ASA), can be expressed algorithmically as:

1. Initialise the contents of all memory locations in all nodes to the value u . Set a count variable $c_1 = 0$.
2. Choose an input pattern and desired output pair. Set a count variable $c_2 = 0$.
 - 2.1. If $c_1 < c_{end}$, do $c_1 = c_1 + 1$.
 - 2.2. If $c_1 = c_{end}$, stop.
3. Present the input pattern to the input of the pyramid and allow values to propagate through the network.
4. Compare the value y , outputted at the network’s output PLN, with the desired output value d . Compute the error signal $\delta = y - d$,
 - 4.1. If, $\delta = 0$, store the value outputted by each node of the pyramid as the new contents of the memory location addressed by the inputs to that node. This ensures that current, successful behaviour will be repeated. Then go to step 2.
 - 4.2. If, $\delta \neq 0$, do $c_2 = c_2 + 1$.
 - 4.2.1. If $c_2 < c_{reset}$, go to step 3.
 - 4.2.2. If $c_2 = c_{reset}$, reset, for all nodes, the contents of the addressed memory location, to the value u . Set $c_1 = 0$ and then go to step 2.

The value of the parameter c_{end} is set in such a way that it guarantees satisfactory responses to all patterns in the training set. The value of the parameter c_{reset} is set in such a way that it allows most possible ‘addressing paths’ to be explored by the algorithm. Extensions to the ASA have been introduced by Myers to make it applicable to MPLN pyramids. In [Mye88b], an incremental reinforcement training algorithm for PLN nodes with more than 3 stored values is developed. This is done in the context of exploratory learning, for which the standard training and use phases of the processing are replaced by a single ‘exploratory’ phase [Mye90].

In [A-Ala90], a similar algorithm to ASA is presented, referred to as Al-Alawi algorithm (AAA) [PenS91]. In AAA, the search in state space corresponding to loop (3 $\mapsto$ 4 $\mapsto$ 4.2 $\mapsto$ 4.2.1 $\mapsto$ 3) of ASA, is performed deterministically instead of stochastically. The algorithm can also be extended to MPLN pyramids. In [PenS91], it is argued that although the AAA can get stuck at local maxima, it is not likely to be a problem in practice. Also, when compared with ASA, the AAA achieves a reduction of the number of iterations per pattern presentation, at the expense of an increase in the required number of presentations of the training set.

The Noise Test Algorithm (NTA), proposed by Kan [Kan86], is similar to the ASA

[Ale89a] but operates in batch mode. During one iteration of the algorithm, all possible input-output pairs, or instances, are presented to the network. The instance that gets learned is the one that produced the highest error rate [Ale86].

In Section 4.4, as a thesis contribution, the dynamics of learning in a PLN pyramid with parameters $N = D = 2$, trained with ASA, are studied and the transition probabilities of the pyramid's internal state during training are calculated.

2.3.5.3 Functional capacity

The functional capacity, or functionality, of a neural node or network is defined as the number of distinct functions that this node or network can perform in response to training [Ale88]. The first work on the functional capacity of a pyramidal feed-forward neural network was carried out by Myers [Mye88a] who found a recursive polynomial formula for the case $N = 2$, with arbitrary D . Denoting by $\Gamma(N, D)$ the functional capacity of a pyramid with parameters N and D , the formula is given by

$$\Gamma(2, D) = 2 + 2p + 2p^2 + \frac{p^2}{2},$$

with $p = \Gamma(2, D-1) - 2$ and $\Gamma(2, 0) = 4$. In [Fon88], the functional properties of the network are analysed, by exhaustive search, for $N = D = 2$. A general formula for arbitrary N and D was stated independently by Aleksander¹, Redgers [Red88], and Al-Alawi [A-Ala89] and the concept of *non-trivial partitions* of patterns at the node inputs introduced. The following double recursive formula was obtained:

$$\Gamma(N, D) = \sum_{i=0}^N \binom{N}{i} \phi(N-i) \left[\frac{\Gamma(N, D-1) - 2}{2} \right]^{N-i}, \text{ with } \Gamma(N, 1) = 2^{2^N};$$

$$\phi(k) = 2^{2^k} - \sum_{j=1}^k \binom{k}{j} \phi(k-j), \text{ with } \phi(0) = 2,$$

in which $\phi(k)$ represents the number of distinct Boolean functions which depend exactly on k *particular* variables.

¹ Personal communication. Oct. 1988.

Al-Alawi extended this formula to include the case of pyramids with variable connectivity N at each layer. In [A-Ala90], the functionality of the pyramid is compared with that of the WST neuron model [Mur65] and it is found that the pyramid has a much greater functional capacity than the WST node. An algorithm is also given to check whether a particular function can or cannot be achieved by the pyramidal node [A-Ala90].

The results mentioned above give formulae that are recursive in both the number of layers D of the network and the number of inputs N to each node. In Chapter 4, as a contribution to this thesis, an *exact* and *non-recursive* formula is derived for the functional capacity of a D -layer pyramidal network with 2-input nodes ($N = 2$), and it is shown that the functional capacity of the network grows as 6^{2D} .

2.3.5.4. Generalisation performance

In [Sha89a], it is argued that the pyramid node may have a sub-node structure which may be too restrictive from a computational point of view. Indeed, there is only one path from any input bit to the single node output. If any PLN implements a function which only depends on a few of its inputs, then all nodes below and which connect to the other irrelevant inputs of that particular PLN will be ignored. This suggests a bad performance of the pyramidal node for higher-order problems. For the parity checking problem, for example, one finds that most changes to stored values result in no change to the error measure, and the training amounts therefore to a random walk. Learning time is found to be exponential in the number of inputs to the pyramid [Sha89a] [Ale89a] [Mye90]. This also confirms the complexity results obtained by Judd on the NP-completeness of learning in neural networks of depth greater than two [Jud90]. In [Mye90], it is argued that PLN networks and in particular pyramids learn more quickly than WST nodes but only in the case of small networks.

In [PenS92], a simple model of the generalisation process, based on information-theoretic considerations [Den87], is used to assess how accurately a pyramid network can classify unseen patterns after having been trained on a number of training patterns and assuming that both training and test patterns belong to a function that the pyramid is able to perform. This provides an estimation of the generalisation error as a function of the network connectivity and the number of training examples. This estimation, however, is based on very unrealistic assumptions. For example, it is assumed that, throughout training, after a new training pattern has been learned, *half* of the remaining functions (compatible with the training patterns learned so far) will be compatible with the next training pattern. Furthermore, it is assumed that all functions that the network can perform are equi-probable. Despite these assumptions, the conclusion holds that, because the number of functions the pyramid can represent is many orders of magnitude smaller than

the total number of possible Boolean functions, in most cases, the network is equally likely to respond correctly or incorrectly to some pattern unseen before.

In Section 5.2, it is shown that pyramids are unsuitable for training in a Kohonen-type network under an unsupervised training scheme, due to their generalisation characteristics.

2.3.6. The continuously-valued discriminator node

In chapter 5, as a thesis contribution, the continuously-valued discriminator node, or C-discriminator (CDN), is introduced as an extension to the discriminator node described in Section 2.3.3. The memory locations in a C-discriminator can hold a continuous range of values between 0 and 1. The RAM sub-nodes of a C-discriminator will be referred to as Continuously-valued RAMs (C-RAM).

2.3.7. The generalising random access memory node

2.3.7.1. The ideal artificial neuron

In [Ale90a], an *Ideal Artificial Neuron* (IAN) is defined. The model has the following properties:

1. The IAN records the appropriate response to all patterns in the training set (during a learning period). It is capable of producing these responses at its output when addressed by all such patterns.
2. Given an input due to a state not in the training set, the IAN must produce a response as the most similar pattern in the training set.
3. States which lead to input patterns that do not have a clear, single, most similar element of the training set cause the output to be 0 or 1 with equal probability.

The Generalising Random Access Memory model (GRAM) is a physical embodiment of the IAN and is reviewed in the next sub-section.

2.3.7.2. The GRAM model

A Generalising random access memory (GRAM) is essentially a 3-value PLN, augmented with an internal generalisation mechanism called *spreading*. The GRAM operates in 3 phases [Ale90a]. During the *training* phase, the node records addresses with their required response. Then comes a *spreading* phase, during which memory locations not addressed during the training phase, are affected by the use of a suitable *spreading algorithm*. Finally, during the use or *operating* phase, the node makes use of its stored information.

The aim of the spreading phase is to diffuse the information acquired by the node during training in order to be able to generalise its knowledge to "unseen" patterns.

Different spreading procedures can be used. For example, in Chapter 7, a spreading algorithm is considered which gives *full generalisation*. It proceeds as follows. All memory locations in a node are considered in turn. If a memory location contains 1 or 0, it is left unchanged. If it contains u , it is set to 1 (or 0) only if its address is closest in Hamming distance to the address of a memory location storing a 1 (or 0); otherwise it is left unchanged. After this operation, the node is said to have full generalisation, that is, unless there is a contradiction, all memory locations in the node are set either to 1 or 0.

An implementation of the GRAM, called Virtual RAM (V-RAM), was proposed in [Mrs93], which consists of only allocating memory space for the training patterns and not for the memory locations which are not addressed during training. During the operating phase, the node responses to inputs not encountered during training are determined by calculation. It is usual, for example, that the output corresponding to the training pattern closest in Hamming distance to the current input pattern is outputted. The VRAM implementation technique leads, in general, to a much lower memory requirement than with a fully-implemented GRAM. However, the advantage of a gain in required memory has to be balanced with an increase in response time during the operating phase.

2.3.7.3. Best matching and diffusion algorithm

The GRAM model is closely related to the general problem of *best matching*. In [Kan88], the problem of best matching is defined as that of finding the element of the network memory closest, in Hamming distance, to the input pattern. In [MinP69], the problem of best matching is also discussed and the learning of a training set L is defined as consisting of storing at each address of the memory the pattern of L which matches the address the best. Retrieval is then reduced to reading just one word from the location addressed by the test pattern. In [Kan88], a filing scheme is proposed for Minski & Papert's algorithm, which consists of:

1. storing the training patterns at locations addressed by the patterns themselves,
2. computing, for each training pattern T_i , all the patterns that are one bit away from T_i ,
3. writing T_i in locations addressed by these distance-1 patterns, unless a location is already occupied.
4. repeating for distances 2, 3, and so on, until all the memory is filled.

Kanerva defines the Best Match Machine (BMM) in which finding the best match for a test pattern U , consists of placing U in the address register and finding the least distance for which there is an occupied location. Only one location is needed for each training pattern and none of the unoccupied locations need be present.

The work reported in [WonS89] relates closely to the concept of spreading algorithm introduced in the GRAM model. In [WonS89], a RAM-based auto-associative network

was studied in the case of low connectivity for the storage of uncorrelated patterns. The learning was based on a noise training algorithm, which was found to be equivalent to the so called *proximity rules* [WonS89], in the limit of low training noise level. A *majority rule* was then derived from the proximity rules, as an alternative way of training the network. The majority rule gives a higher storage capacity than any noise-trained algorithm. It proceeds as follows. The contents of a location follows the pattern which is its nearest neighbour, if there is only one such; the contents follows the majority if there is more than one nearest neighbour pattern; if there is no majority, the location is randomly filled with 0 or 1 with equal probability. This leads to a new way of training the network called *diffusion algorithm*:

1. Each pattern is stored at the correct memory location of each node. If more than one pattern addresses a location, its contents is determined by the majority rule.
2. The contents of these locations are copied to their nearest neighbours which are still unaddressed. If more than one pattern addresses a location, its contents is determined by the majority rule.
3. Point 2 is continued until all the memory locations are filled.

2.3.8. The dynamically generalising random access memory node

In Chapter 8, as a contribution to this thesis, the Dynamically Generalising Random Access Memory (DGRAM) is introduced. The DGRAM is derived from the GRAM. The training and spreading phases of the GRAM are replaced by a single *learning* phase [Nto93]. The DGRAM is able to store and spread patterns, through a dynamical process involving interactions between each memory location, its immediate neighbours and external signals. The DGRAM exhibits very desirable properties, compared with those of the GRAM. First, after the initial trained patterns have spread throughout the memory space, additional patterns can still be stored in the DGRAM. Secondly, it is possible to distinguish between trained and spread patterns. And finally, a trained pattern and its associated spread patterns can be removed without affecting the rest of the stored patterns.

2.3.9. Other weightless node models

In [GorT88], a noisy neuron model is developed, which includes many physiologically realistic features, and is shown to be equivalent to a network of noisy RAMs. This leads to the definition of the Probabilistic RAM (P-RAM) [GorT90] [GorT93]. The P-RAM is basically an extension of the PLN, in which each memory location can store a continuous value in the range $[0, 1]$ and which outputs a spike-train signal whose frequency is related to the contents of the addressed memory location. The memory contents itself is a function of the frequency with which the memory locations have been accessed over some number

of previous time steps. In [Mye90], attention is drawn to the similarity between the P-RAM and MPLN weightless neuron models. In [Gur89], a similar model to the P-RAM, called S-model, is developed (S stands for 'stochastic'). The S-model is derived from an analog extension of the RAM node called A-model (A stands for 'analog').

In [FilBF90], the Goal Seeking Neuron model (GSN) is proposed as a deterministic version of the PLN. The GSN can also store values from the set $\{0, 1, u\}$, but in contrast with the PLN, it also accepts these values as inputs and generate them as output. If the input to a GSN contains u values, a series of locations in the node are accessed whose addresses are obtained by replacing the u values in the input by the values 1 and 0. So, if d inputs are at u , 2^d locations are visited in the node. GSN nodes are used in similar architectures as PLN nodes, the pyramidal network being a popular one [FilFB92] [CarFBF91]. There are 3 phases of processing in GSN networks. During a *validation* phase, some strategy is employed to decide if the requested input-output mapping is possible. During a *learning* phase, changes to the GSN contents are performed. During the *recall* phase, if several locations are accessed in a node, the output is decided by majority rule. Improvements to GSN networks have been proposed which relate to output encoding and selection of initial topology [MarA93].

In [Kru91], the Linear Interpolating Boolean Node (LIBN) is introduced. It is a RAM node with both inputs and outputs taking the binary values -1 or 1 and which is trained by a gradient descent method. During training, the LIBN output function is expressed as a linear interpolation between the memory location contents. This continuous nature of the processing is only necessary during training. After training is completed, the node strictly operates in a RAM mode in which the output is obtained by addressing a memory location with some input pattern. It is the linear interpolation which differentiates the LIBN from the Derived Probabilistic Boolean Node (DPBN), introduced in [Mar89] as an extension to the PLN and which is also trained by an adaptation of the Error Back Propagation (EBP) algorithm.

In [Vid88], the concept of Dynamically Programmable Logic Modules (DPLM) is introduced. These are basically non-stochastic RAMs, arranged in tree structures and trained by truth table decomposition. These are demonstrated in the context of moving edge detection, tracking and detection of pattern of some minimum size in the input data.

In [Moo89], a Memory-Based Learning Controller (MBLC) is used in the context of robot control. The MBLC involves explicitly remembering every experience in the lifetime of the robot. Then, responses to new inputs are provided either by Nearest-Neighbour Methods (NNM) or by Local Weighted Regression (LWR). Learning is very fast and the predictions can be made fast enough for real time control. The important point is that memorising all previous experiences permits a variety of forms of mental simulation and self-checking. The resulting system is one which improves very quickly when learning simple problems, yet which does not get stuck in false minima when learning hard ones.

In [ZhaC90], the Lagrange Programming Neural Network (LPNN) is introduced. It is a neural network designed for non-linear programming. The LPNN has been demonstrated in the context of maximum entropy image restoration [ZhaC91], which is shown to be equivalent to an optimisation problem in which one or more expressions must be minimised, subject to some conditions. A *dynamical* system of equations can be formulated, whose equilibrium point provides a Lagrange solution to the optimisation problem. The neural nature of the processing follows from the implementation of the dynamical process as a connectionist VLSI circuit. The LPNN is somewhat different from the RAM-based systems reviewed earlier in this chapter. However, it is justified to review this neural model in the context of WANNs, as neither weights nor thresholds are necessary to describe its behaviour. The only adaptive parameters of the system are held in memory elements, referred to as *parameter neurons*, while the dynamical process is performed by so called *variable* and *Lagrange neurons* [ZhaC91].

2.4. Properties of weightless neural nodes

2.4.1. Introduction

Section 2.3 has reviewed a series of weightless node models, which can be characterised by a number of the following 4 properties: the training is performed by node loading; the generalisation is partly or totally achieved by a spreading mechanism; the generalisation is partly or totally achieved by node decomposition; the node function is allowed to become probabilistic. The 4 properties are examined in more detail in the following sub-sections.

2.4.2. Node loading

Due to the RAM-based nature of all weightless neural nodes, training mainly takes place by storing the correct information at the correct addresses in the nodes. This storage operation has also been called *loading* [Ale90c], by reference to a connectionist learning protocol defined by Judd. In [Jud90], a computational problem, referred to as *loading problem*, is defined as an approximation to the common connectionist notions of learning. *Loading* is defined in the context of supervised learning. Input patterns, the *stimuli*, are presented to a machine paired with their desired output patterns, the *responses*. A *task*, that the system is required to learn, is defined as a set of Stimulus-Response (SR) associations that the machine is required to learn. Learning consists of remembering all the associations presented during the training phase. The retrieval process consists of accepting a stimulus and examining the memory in order to find and output the associated response. Figure 2.5 shows a schematic representation of Judd's learning protocol.

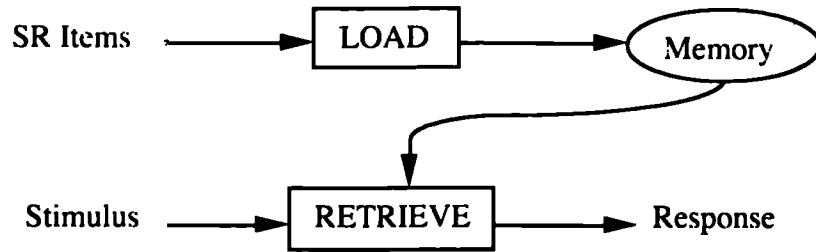


Fig. 2.5: General learning protocol, as defined by Judd.

2.4.3. Generalisation by spreading

In [Ale90b], *spreading* is defined as referring to a process of affecting the contents of storage locations, not addressed during training, by the use of a suitable algorithm which may be implemented on-chip or through appropriate actions in the control machinery. The idea of spreading was first mentioned in [Ale89a] as a means of implementing a PLN's ideal contents, using some machinery internal to the node. Related work can also be found in [Kan88], in which the concept of Best Match Machine is discussed, and also in [WonS89], in which a diffusion algorithm is defined (see Section 2.3.7.3).

Different types of spreading can be distinguished: spreading generated by *noise training* [Ale89a] [WonS89]; *explicit spreading*, as in the C-discriminator training algorithm [Nto90] (see Chapter 5) or as in the GRAM spreading algorithm [Ale89b] (see Section 2.3.7); and *dynamical spreading*, as in the DGRAM learning algorithm [Nto92d] (also in [Nto93] and in detail in Chapter 8, as a contribution to this thesis).

2.4.4. Generalisation by node decomposition

As already exposed in Sections 2.3.3 and 2.3.5, the main reason for node decomposition in WANNs is to allow a measure of control over the generalisation and discriminatory powers of the weightless nodes and networks. A secondary reason for node decomposition is to limit the exponential growth of the required storage with the number of inputs to the nodes. Figure 2.6. shows a schematic representation of the pyramidal and discriminator decompositions of a weightless node.

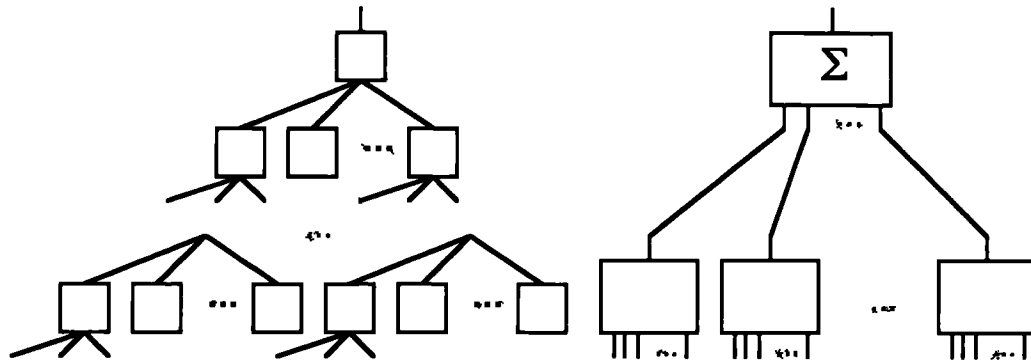


Fig. 2.6: Pyramidal and discriminator decompositions of a weightless node.

2.4.5. Introduction of a probabilistic element

The main reason for the introduction of the u state in the PLN model and the models derived from it, was the need for the implementation of a “don’t know” state. In [KanA89], it is argued that the initialisation of the memory contents to the value u enables the network to start the training in an un-biased state. If the memory contents was preset to deterministic values, this would provide a pre-existing state structure which would need to be disrupted before training to the desired stable states can even begin [Mye90]. Instead, learning in weightless nodes consists of, where possible, adjusting u values only, thus exploring uncommitted regions of function space. Only when the stored knowledge is inconsistent with the current input-output pair does a reset operation take place, which returns the currently addressed locations to the unbiased state.

It was observed (e.g., [Ale73]) that in recurrent networks, the network tends to enter limit cycles if an probabilistic element is not introduced. The introduction of the u state reduces the possibility of cyclic dynamics which prevent the retrieval of correct patterns². This also points out to the importance of having a certain amount of noise present in the system.

Another reason for the introduction of the probabilistic element u is to enable the training of hidden layers in multi-layer nodes or networks (see Section 2.3.5.2).

In [WonS89], it is argued that, from the point of view of storage capacity, the introduction of the state u does not improve the performance of the network. This question

² It is argued in [WonS89] that for networks of low connectivity, the possibility of cyclic dynamics is irrelevant

of storage capacity was discussed in the particular framework of a sparsely inter-connected recurrent PLN network, with a low training noise procedure. Applying a majority rule, for the particular network configuration studied, Wong & Sherrington argue that it is always better to fill a memory location with the majority bit, if there is any, than with a μ value. These criticisms of the μ value are made in the particular context of the training with noise algorithm. In that case, inevitably, some memory locations which contained the correct bit are filled with the μ value, and the storage capacity of the network therefore deteriorates. This will no longer be the case in the General Neural Unit (GNU) network with GRAM nodes (see Section 2.5.3.3).

2.5. Weightless neural networks

2.5.1. Network structure levels

In [Ale83b] (also in [Ale84a]), several levels of weightless network structures are distinguished. These are represented in Figure 2.7.

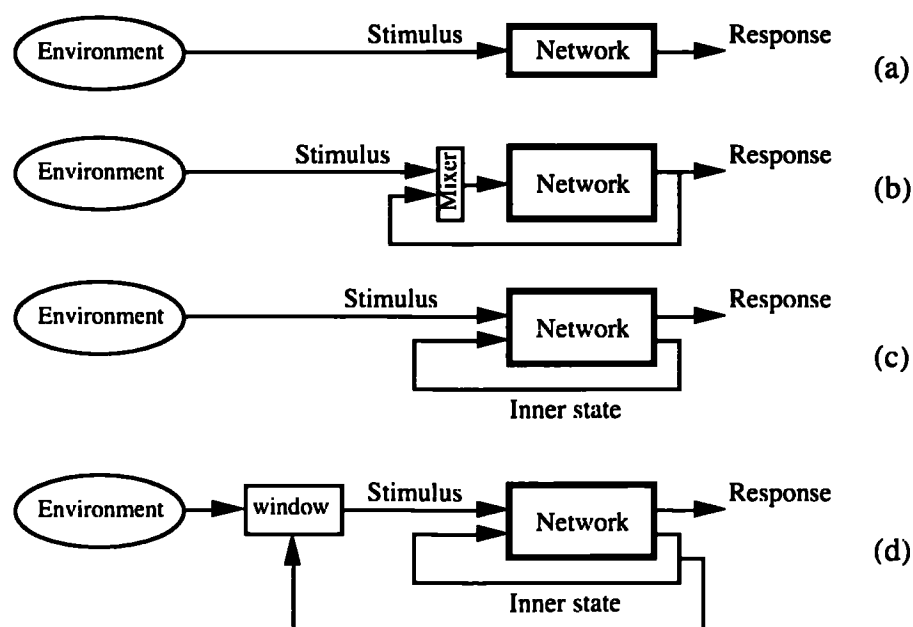


Fig. 2.7: Network structure levels: (a) level-0; (b) level-1; (c) level-2; (d) level-3.

Level-0 structures are basically feed-forward networks. These networks have been shown to exhibit generalisation properties, by acceptance of diversity with respect to the stimuli coming from the environment [Ale83b]. Networks with levels 1 to 3 are structures which all have various degrees of feed-back.

In level-1 structures, the output of the network is fed-back and mixed to the incoming stimulus from the environment. This type of feed-back has a role of amplification of

confidence. Level-1 structures exhibit a short term memory of the last response of the network.

In level-2 structures, the input field to the network is split between an external input field and a feed-back input field. The output of the network can directly be used as the feed-back input field, but generally a separate feed-back output function is provided. Level-2 network structures are also referred to as *recurrent networks* [Jor88]. This type of network is able to store *inner state* images or pattern prototypes. The system, after having been trained on these prototypes, is then able to respond to variants of input images by settling into one of these inner states. Prototype inner state representations have also been referred to as *iconic* states [AleM93] (also in [AleEP93] or [NtoS94]). Level-2 structures have been shown to exhibit a number of associative and temporal network behaviours. They can be trained to respond to a static stimulus by a static response or a sequence of responses. Correspondingly, they can be trained to respond to a sequence of stimuli by a static response or by a sequence of responses, displaying therefore properties such as sequence sensitivity.

In level-3 structures, two levels of feed-back can be distinguished. The first type of feed-back is the one found in level-2 structures. The second type of feed-back takes place between the inner state and the external input, creating therefore an *attention mechanism* in the network, by which the inner state of the network influences the next stimuli arriving at the input of the network. This type of feed-back can be implemented by a windowing system, with zoom, panning and scrolling mechanisms (e.g., [Ree73], [Dow75], [Ale84b], [AleEP93] or [NtoS94]). The window position control can be done either by pre-programmed scan [Ree73], by small increments [Dow75] or, as suggested in [Ale84b], by *saccadic jumps* [NtoS94]. Saccadic jumps are context-dependent and constitute a powerful tool, not only in object labelling in multi-object scenes, but also in the analysis of similarly shaped objects which differ in local detail [Ale84b].

2.5.2. Feed-forward weightless networks

2.5.2.1. Introduction

In this Section, two feed-forward WANNs are reviewed: the single layer discriminator network and the advanced distributed associative memory, which is a two-layer discriminator-based WANN. Studies using feed-forward WANNs with pyramidal nodes can be found in [Mye90]. Also, it is worth noting that single discriminator nodes or pyramidal nodes are sometimes referred to as *networks*, due to their decomposed structure which can be interpreted as that of a network of RAM or PLN nodes. In Chapter 4, in which feed-forward pyramidal networks are studied, as a thesis contribution, it is the network interpretation of a pyramidal node which is used.

2.5.2.2. The single layer discriminator network

2.5.2.2.1. Description of the network

In Section 2.3.3, a single discriminator node was presented. In classification problems involving more than one class of patterns, a network made of one layer of discriminators with parameters K , N is used [AleS79]. A number z of discriminators is assumed, corresponding to z different classes $X_1, X_2, \dots, X_z$, with each class of patterns to be classified being assigned to a different discriminator. Figure 2.8. shows a schematic representation of the overall recognition system.

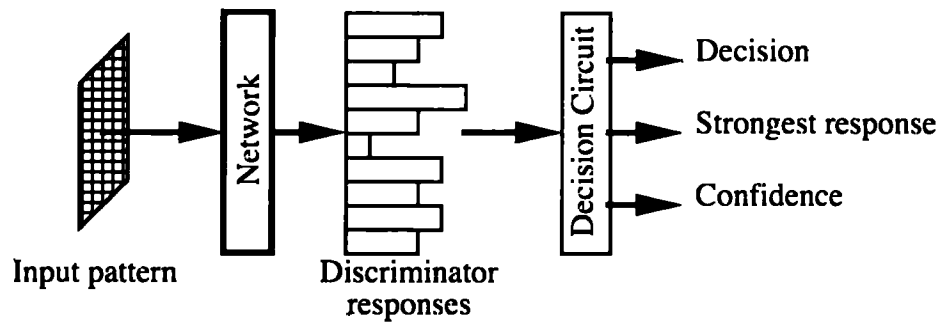


Fig. 2.8: One-layer feed-forward discriminator network.

The input to the network consists of an array of m input pixels. A random mapping is applied between the input pixels and the K exclusive N -tuples in each discriminator:

“A random map is chosen, because sampling points distributed throughout the pattern matrix are more likely to detect global features than an ordered map, which in a single-layer system is only sensitive to local features” [AleS79].

The outputs of the DNs feed in an overall *decision circuit* which assigns the classification of an unknown pattern to that DN which has the strongest response. It is worth noting that with the use of a decision circuit, the predetermination of output-decision thresholds is avoided.

2.5.2.2.2. Learning and generalisation

During a learning phase, each discriminator is trained to a set of patterns belonging to the same class. For each training pattern, a 1 is stored in the RAMs of the discriminator corresponding to the pattern class, at the memory locations addressed by the input pattern. For each class of patterns, numerous examples of that class, differing in shape or position

or both, can be learned. A key point of this training procedure, noted by Bledsoe and Browning, is that the very shape of a pattern forbids certain memory locations from being addressed and thus prevents the logic from saturating too quickly.

During the recognition phase, an unknown pattern is scored against the discriminator responses and identified as a member of the class corresponding to the discriminator which scores highest [Ste62]. Different discriminators are trained to different patterns, thus setting different memory location contents to 1 in each discriminator. Therefore, the responses of the different discriminators to an unknown pattern are different.

Control can be gained over the generalisation of the network by insisting that there will be a minimum difference between the responses of the discriminators responding highest and second highest, r_1 and r_2 . This enables the control the reject class size and leads to the definition of a *confidence* parameter C defined as ([Ale83b]):

$$C = \frac{r_1 - r_2}{r_1},$$

which can be approximated, using (2.6), by

$$C \approx 1 - \left(\frac{\omega_2}{\omega_1}\right)^N,$$

in which ω_j represents the overlap between the input pattern and the closest pattern in the training set for discriminator j . The “don’t know” category is particularly important. If the confidence C is smaller than some reject threshold ρ (i.e. $C < \rho$), the input pattern is classified as “don’t know”. It should be noted that here the “don’t know” concept is implemented at the network level. It is interesting to put this in contrast with PLN networks in which the “don’t know” concept is implemented at the node level in the form of the u contents.

It has been found empirically that for a given size t of training set, there is an optimum value N which will give maximum performance, smaller values of N causing over-generalisation and larger ones under-generalisation [AleS79].

The number of training patterns t is a parameter of the system which has similar properties to N ; small training sets have an effect similar to small N -tuple samples. When t becomes large, the network memory starts saturating with 1 values. This *saturation* characteristic is particular to the particular type of training considered, in which the contents of memory locations can only be set to 1 but never reset to 0. Steck notes;

“Usually in sampling problems the more observations one has, the more information one has about the statistical population in question. Here the opposite is true: beyond a certain point the more observations one has, the *less* information one has about the population of

patterns" [Ste62].

The possibility of non-exclusive N -tuple sampling of the input pattern is considered [BleB59] (also in [Ste62]). In [AleS79], a *coverage* parameter V is then defined as:

$$V = \frac{K \cdot N}{m},$$

It has been shown experimentally that some improvement in the percent of correctly recognised patterns occurs when $V > 1$ (oversampling) but that the improvement is marginal [BleB59] and that, as V is increased, the performance of the system rapidly reaches a asymptotic value. Also, an increase in coverage increases the amount of memory and computing time necessary to implement the network. There is therefore no real advantage in non-exclusive grouping. In [AllBJ89], oversampling is used, in the context of an N -tuple sampling-based Kohonen network, to improve reconstruction of corrupted input vectors during the use phase.

Finally, it is also interesting to mention the experimental results obtained in [AleW85] and which show that a discriminator network trained on an edge-detection problem performs at least as well as Sobel and Laplace transforms on a binary image and is less sensitive to noise distortion [Mye90].

2.5.2.2.3. Steck's stochastic model

Using a stochastic model, Steck [Ste62] gives, for the discriminator network, the probabilities of successful recognition as a function of the structural parameters z , K , N of the network and of pattern variability parameters π and ν of addressing a 1 at a memory location when it is wanted or unwanted, respectively. Steck [Ste62] gives a semi-empirical method for estimating the variability parameters π and ν as functions of the parameters N and t , where t represents the number of experiences of the learning process for each class.

A pattern x_p , presented to the network, selects K memory locations in each of the z discriminators. The notation f_{ij} is used to denote the value addressed in the i -th RAM of the j -th discriminator.

A number of assumptions are made on the independence and distribution of the addressed values f_{ij} :

- 1) The f_{ij} are assumed to be independent. The assumption is reasonable since no two RAMs are looking at the same part of the input pattern. In reality, correlations between N -tuples exist but are small if the within-class variability is considered as resulting only from uniformly distributed noise and if causes producing high correlations between patterns (i.e., shift, scaling, ...) are neglected.
- 2) The values $f_{1p}, \dots, f_{Kp}$, addressed in discriminator p trained on patterns belonging to



the same class as input pattern x_p , are all assumed to have identical binomial distributions $B(1, \pi)$ ([GriS82]). This assumption is unrealistic since some RAMs are better than others for the recognition of a pattern within discriminator p . However, the assumption is reasonable because the variations are averaged out.

- 3) The values f_{ij} , with $i = 1, \dots, K$, $j = 1, \dots, z$ and $j \neq p$, have all identical binomial distributions $B(1, \nu)$. This assumption is both unrealistic, since some classes of patterns look more alike than some others (effect of the among-class variability), and unreasonable, since the variations in among-class variability are not averaged out.

So, a pattern x_p , presented to the network, yields a set of responses $r_{p1}, r_{p2}, \dots, r_{pz}$ to which correspond a set of random variables $R_{p1}, R_{p2}, \dots, R_{pz}$, with R_{pp} a random variable with distribution $B(K, \pi)$ and R_{pj} ($j \neq p$) $z-1$ random variables with distribution $B(K, \nu)$. Summing over all values of the R_{pj} , the probability of correct recognition $P_c(x_p)$ of input pattern x_p can therefore be expressed as:

$$\begin{aligned}
 P_c(x_p) &= \sum_{k=1}^K \Pr\left\{\left(R_{pp} = k\right) \wedge \left(\max_{j \neq p} R_{pj} < k\right)\right\} \\
 &= \sum_{k=1}^K \Pr(R_{pp} = k) \cdot \left[\Pr(R_{pj} < k \mid j \neq p)\right]^{z-1} \\
 &= \sum_{k=1}^K \binom{K}{k} \cdot \pi^k \cdot (1-\pi)^{K-k} \cdot \left[\sum_{l=0}^{k-1} \binom{K}{l} \cdot (1-\nu)^K \cdot \nu^l\right]^{z-1}, \quad (2.10)
 \end{aligned}$$

2.5.2.3. The advanced distributed associative memory network

The Advanced Distributed Associative Memory (ADAM) is a 2-layer discriminator-based network. The system works as a hetero-associator which accepts as input distorted versions of previously stored patterns and retrieves at the output their associated stored patterns. There is no hidden layer in the network. The output of the first layer forms a class pattern, of length Q and with q -point coding (q randomly chosen bits of the pattern are set to 1 and the rest is set to 0). The class pattern is chosen, usually at random, during training, for each association trained in the network. The input pattern is associated with the class pattern in the first layer of the network and the class pattern is associated with the output pattern in the second layer of the network.

The first layer is made of an array of Q discriminator nodes (DN) with parameters K , N (see Section 2.3.3). Each discriminator receives the totality of the input pattern. The mapping, usually random, of the input pattern is identical for each discriminator. The training of an input pattern in the first layer of the network is done by setting to 1 the

contents of addressed memory locations in the discriminators which are required to produce a 1 at the corresponding bit of the class pattern. During recall, when an unknown pattern is presented for association, the response of all discriminators is thresholded, such that the q discriminators outputting highest produce a thresholded 1 and the rest a thresholded 0. This type of thresholding is referred to as q -point thresholding. The test patterns are expected to be distorted versions of the training patterns and therefore the first layer of ADAM is required to exhibit generalisation properties. The generalisation ability of the first layer is a consequence of the generalisation properties of discriminator nodes and of the generalisation provided by the q -point thresholding operation.

The second layer of the network is made of one array of linear unit elements (LU) with *binary* weights. Its role is to associate the class vector used to train the first layer with the required output pattern. The second layer of the network needs no ability to recall on incomplete data. Indeed, it is expected that the input, i.e. the class pattern retrieved by the first processing layer will be the same as the one taught during training [Aus86]. This lack of required generalisation in the second layer justifies the choice of LUs with binary weights instead of DNs. The threshold used at the outputs of the LUs is the same for all units and set to a value q , justified by the q -point coding of the class input patterns to the second layer.

The parameters of the ADAM system are K , N , Q and q . A schematic representation of the ADAM system is given in Figure 2.9.

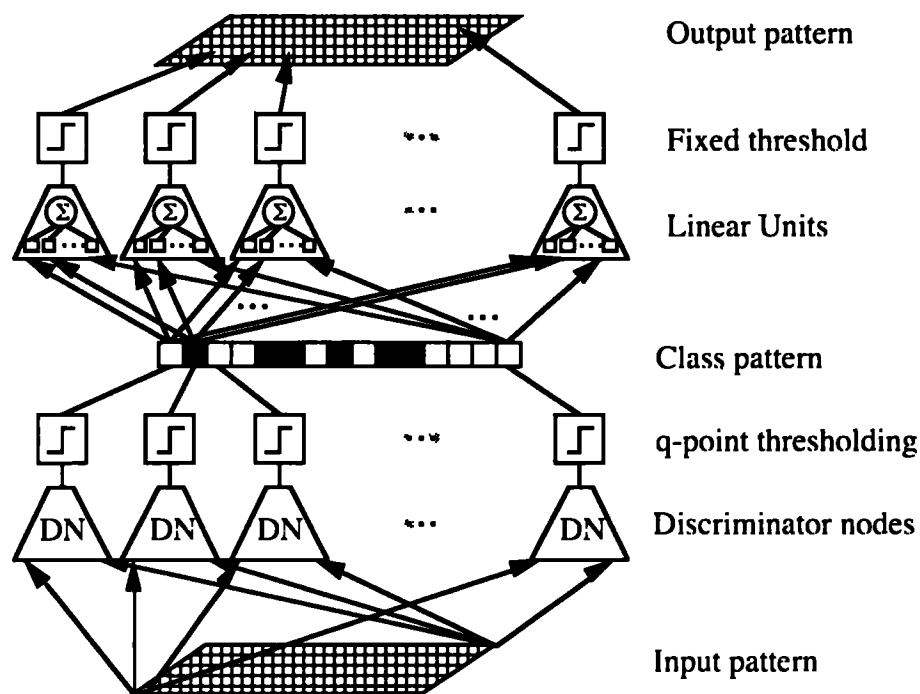


Fig. 2.9: A schematic representation of the ADAM system.

The main cause of recall errors in both layers is the saturation of the network. The effect of saturation in the second layer results in erroneous bits appearing in the output pattern. This also affects the first layer and diminishes its ability to recognise incomplete input patterns. The errors occurring at the output of the first layer are due to extra discriminators outputting a maximum value K , as a consequence of their saturation. In that case, the q -point thresholding will produce errors. The storage capacity of the network can be estimated based on the calculation of the probability that a discriminator outputs wrongly a maximum response K . The storage capacity is then defined as the number of patterns that can be loaded before an error bit appears at the class pattern, on average over a number of test pattern presentations. It is shown in [Aus86] (also in [AusS87]) that this number S of patterns can be expressed as a function of the network parameters:

$$S = \log\left(1 - \sqrt[q]{Q^{-1}}\right) \cdot \log^{-1}\left(1 - \frac{q}{Q} \cdot 2^{-N}\right).$$

The ADAM network has been used in scene analysis applications, for which objects to be recalled may be occluded by other objects in the scene [AusS87]. The system can also be augmented with a pre-processing stage to provide it with rotation and scale invariance capabilities [Aus89]. Grey level inputs are also possible, using a suitable coding of the input patterns [Aus88]. Other improvements include the possibility of storing multi-valued data in the memory and also an automatic selection of the class patterns by a Kohonen-type algorithm [Aus93].

2.5.3. Recurrent weightless networks for associative memory

2.5.3.1. The sparsely-connected auto-associative PLN network

In [WonS88] (also in [WonS89]), results on the study of randomly and *sparsely* connected auto-associative PLN networks are reported. For sparsely connected networks, the number F of inputs per node is much smaller than the number K of nodes in the network ($F \ll K$),

The method of investigation used is Statistical Mechanics (e.g. [CamSW89]), which deals with storing and retrieving as energy minimisation problems. In studies on storage capacity, one typically fixes the set of stored patterns and looks for parameter configurations (the parameters being either weights or memory contents) that minimise an energy function. The study of retrieving can be viewed as the inverse of storing, in which one typically fixes the parameters and looks for neuronal states that minimise an energy function [WonS91].

The study by Wong & Sherrington is only concerned with the storage and retrieval

properties of the network and not with training times and efficient learning algorithms. Most of the results obtained are asymptotic in the number of training steps. They are calculated for a network reaching an equilibrium state after an infinite number of training pattern presentations.

The associative properties of the PLN network are obtained by a *training-with-noise* algorithm (first proposed by Aleksander in [Ale89a]). According to this algorithm, all memory locations are first initialised to the value u . Then, during each training step, an example pattern is presented to the input of the nodes. Addressed memory locations containing *correct* bits are left unchanged; addressed memory locations containing u are filled with the correct bit; and addressed memory locations containing the incorrect bit are reset to u . The main training parameter is a *training noise level* d . A noisy version of a pattern required to become a stable state of the network is presented many times to the network. Each noisy instance is obtained from the original pattern with a fraction d of its bits set at random.

For a large number S of stored patterns T_i , with $i = 1, 2, \dots, S$, the mean field approximation holds and the dynamics of the system can be described by mean field parameters x_i describing the distance of the network state to stored pattern T_i . The dynamics of the network are determined by retrieval equations $f_i(x_i)$ such that:

$$x_i(t+1) = f_i(x_i(t)).$$

It is the stability of the fixed points of the retrieval equations that is of interest. A stable fixed point at or near a stored pattern (i.e. $x_i = 0$) means that the network has associative memory, i.e. starting with an initial configuration which has only partial agreement with a stored pattern, it eventually retrieves the pattern [WonS89]. For example, the retrieval equation corresponding to the storage of one pattern, with no added noise, is [Ale87] (also in [WonS88]):

$$x(t+1) = \frac{1}{2} \left(1 - [1 - x(t)]^F \right).$$

It can be shown that, for $F > 2$, the network has almost no associative memory. However, if t additional training steps are performed, with noisy versions of the stored pattern, the retrieval equation becomes:

$$x(t+1) = \frac{1}{2} \sum_{j=1}^F \binom{F}{j} \cdot [1 - d^j (1 - d)^{F-j}]^t \cdot x^j \cdot (1 - x)^{F-j}$$

and the fixed point at $x = 0$ changes from unstable to stable, enabling therefore associative

memory.

The main result obtained by Wong and Sherrington is the storage capacity of the network, in the limit of *very low training noise* and for *uncorrelated patterns*. In that particular case, the network is shown to have a maximum storage capacity

$$S = \frac{2^{F+1}}{F^2}, \quad (2.11)$$

which corresponds to an average of one nearest-neighbouring pattern stored at Hamming distances 2 on each node [WonS89]. Moreover, it is shown that low training noise levels is the best strategy for high storage and that, in that case, the training-with-noise algorithm is equivalent to so-called *proximity rules* (see also Section 2.3.7.3 for a further discussion of the proximity rules and resulting *diffusion algorithm*, in the framework of spreading methods).

In [WonS89], the characteristics of the PLN network are also compared with those of a dilute asymmetric Hopfield-Little network having the same topology as the PLN network but with synaptic storage. The storage capacity for the Hopfield network is only ([DerGZ87])

$$S = \frac{2F}{\pi} \approx 0.64F,$$

which is much smaller than (2.11) for large F . It turns out that, in common with the Hopfield-Little network, the PLN network exhibits a memory threshold, beneath which it can store with only small errors, but at which it experiences a memory catastrophe [WonS89]. When the extent of the basins of attraction are considered, the PLN network can be categorised as *short-ranged* whereas the Hopfield-Little network exhibits a *long-ranged* behaviour. In a PLN network, the more patterns are stored the smaller their radius of attraction becomes. This is not the case in the Hopfield-Little network where the basins of attraction are much wider and are independent of the number of stored patterns. As before, these results hold for uncorrelated patterns.

The short-range aspect of the PLN network can be explained by the fact that the information about patterns is stored in localised regions within each node. The advantage is a minimal interference between patterns which are considerably different; the disadvantage is a relatively small size of the basins of attraction.

2.5.3.2. Fully-connected auto-associative weightless networks

2.5.3.2.1. Pure feed-back PLN networks

In [ZhaZZ91] (also in [ZhaZZ92]), a single-layer PLN network with pure feed-back connections is analysed using Markov Chain Theory [CoxM65]. A series of sufficient conditions for the convergence of the PLN network is provided. Other results include the probability that each state in a given network converges to a stable state, a mean value for the number of steps necessary for any state to converge to a set of stable states, and an upper bound for the average number of steps that it takes for any state to converge towards a stable state. The mathematical developments are briefly reviewed in the paragraphs below.

The training set L consists of a number S of training patterns which are to be made stable states in the network. A training set is said to be *compatible* if no training contradictions occur in the nodes of the PLN network. A training contradiction occurs in a PLN when two different values (1 and 0) are required to be stored at the same address. A training pattern is said to be *complete* if for each node in the network, the inputs to the node are required to access a deterministic value (non- u value). In other words, for a training pattern to be complete, for each node, the required output must be available (no hidden units).

The network architecture A is determined by the connections between node outputs and node input lines. The architecture A and the training set L completely determine the behaviour of the network, which is denoted $N(A, L)$. The state of the network at any moment in time is only determined by its state at the previous moment in time and therefore the network behaviour can be studied as a finite Markov chain. Stable and unstable states of the PLN network correspond to a *absorbing* and *transient* states of the Markov chain, respectively.

An important restriction of the analysis carried out here is that A and L are assumed to be consistent, that is, each training pattern results in a stable state of the network. In other words, there are no training contradictions and the Zhang even suggest that the architecture A is chosen such that it is consistent with L .

A network $N(A, L)$ is said to be *convergent* if, starting from any state, it converges after a finite number of time steps to a set of stable states with probability one. In terms of Markov chain theory, a PLN network is convergent if its corresponding Markov chain is absorbing.

The *closure* $C(L)$ of the training set L is defined as the set of all network states accessing trained locations in each node. $C(L)$ is said to be *closed* if all states in $C(L)$ lead to next states which remain in $C(L)$. $C(L)$ is said to be *complete* if $C(L) = \Omega$, where Ω represents the set of all possible network states. $C(L)$ is said to be *saturated* if

$C(L) = L$. Using these definitions, the following *convergence theorem* can then be formulated:

- (a) If $C(L)$ is closed and complete, then the stable state set is a subset of $C(L)$.
- (b) If $C(L)$ is saturated and complete, the network $N(A, L)$ is convergent and the stable state set is L .
- (c) If the network $N(A, L)$ is convergent and $C(L)$ is closed, then the stable state set is a subset of $C(L)$.

In [ZhaZZ92], it is suggested that a simple way to achieve saturation, completeness and closeness of $C(L)$ is to have a fully-connected PLN network. In that case, the output of each node is fed back as input to each node in the network.

Given a convergent PLN network $N(A, L)$, quantitative information can be obtained concerning the number of steps to convergence. The transition matrix $\mathbf{P}$ of the associated Markov chain can be expressed as

$$\mathbf{P} = \begin{matrix} & \begin{matrix} A & T \end{matrix} \\ \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{R} & \mathbf{Q} \end{bmatrix} & \begin{matrix} A \\ T \end{matrix} \end{matrix} \quad (2.12)$$

in which A and T represent the absorbing and transient state sets. $\mathbf{Q}$ represents the transition probabilities between states in T and $\mathbf{R}$ the transition probabilities from T to A . The matrices $\mathbf{N}$ and $\mathbf{V}$ are then constructed such that

$$\mathbf{N} = (n_{ij}) = (\mathbf{I} - \mathbf{Q})^{-1} \quad (2.13)$$

and

$$\mathbf{V} = (v_{ij}) = \mathbf{N} \cdot \mathbf{R}. \quad (2.14)$$

Considering states s_i, s_j, s_k such that $s_i, s_j \in T$ and $s_k \in A$, it can be shown that:

- (a) the probability that s_j converges to s_k is given by v_{ik} ;
- (b) the average number of steps necessary for s_i to converge to a stable state is given by

$$\sum_i n_{ii};$$

- (c) the average number of times that s_i transits to s_j before leaving T is given by n_{ij} .

Furthermore, if the network is made of K PLN elements and $C(L)$ is assumed to be complete and closed, then, starting from any state, the average number of steps for the network to converge, $\langle \tau_{conv} \rangle$, is bounded upwards and it holds that

$$\log \langle \tau_{conv} \rangle \leq 2K.$$

2.5.3.2.2. The GRAM perfect auto-associator network

In [Luc91], a fully-connected GRAM network with feed-back is considered. The network is trained as an auto-associator. Asynchronous dynamics are used. The network is made of one layer of F GRAMs, with F inputs per GRAM.

It is shown that the network can store a maximum of 2^F patterns. Usually, a number S of training patterns is stored in the network, with $S < 2^F$. It is shown experimentally that when an unknown pattern is presented to the network, the system always settles into the training pattern nearest in Hamming distance to the initial pattern. Also, there are no false minima present and the recall of the stored patterns is therefore optimal.

It is interesting to compare Lucy's perfect auto-associator with a Hopfield network in its standard configuration (i.e. fully-connected). Whereas the GRAM network can store a maximum of 2^F arbitrary patterns [Luc91], this number is reduced to about $0.15 \cdot F$ in the case of a Hopfield network storing uncorrelated patterns and where a small noise is permitted [Ama89] [DerGZ87]. If exact recall is required, the Hopfield network can store $\frac{F}{2 \log F}$ patterns [McEPRV87]. So, the sub-linear growth rate of the capacity of the Hopfield net is in sharp contrast with the exponential growth rate of the Lucy network.

In [Cho88] (also in [Cho89]), using information theoretic arguments, the storage capacity of the Kanerva Associative Memory (KAM) [Kan88] is estimated. Considering a KAM with F -bit stored patterns, its associative properties are determined by the variables r_G , r_A and Ξ , respectively representing the radius of the sphere of attraction of a stored pattern, the *access radius* and the *number of memory locations*. Chou shows that the KAM can achieve an exponential growth rate in capacity, with respect to F , if its parameters r_G , r_A and Ξ are set optimally [Cho88]. However, as in the case of the Lucy network, this exponential growth is obtained at the expense of an exponential growth in the implementation cost of the network. For the particular case $r_G = 0$, using a suffix to show the dependency in F , the storage capacity S is shown to be directly proportional to the number of memory locations Ξ_F :

$$S \propto \Xi_F$$

The addressing stage in a KAM essentially encodes an F -bit input address into a Ξ -bit sparse internal representation, with Ξ being permitted to grow exponentially in F . In general, the encoding scheme is said to be sparse when most of the bits of the encoded patterns to be stored are 0 and only a small proportion of the bits are 1. Related work on sparse encoding of patterns can also be found in [Ama89], where a sparsely encoded associative memory is presented. The storage capacity of such a memory is proved to be

$\frac{F^2}{\log F^2}$, which is much larger than the equivalent $0.15 \cdot F$ found for a Hopfield network.

2.5.3.3. The general neural unit network

The General Neural Unit network (GNU) consists of one layer of K GRAM nodes. The inputs to each node consist of N external inputs and F feedback connections from the outputs of the system. Hence, the total number of inputs to a node is given by $N + F$. When $F = 0$, the system's operation is that of a single layer feed-forward network. In the case $N = 0$ and $F \neq 0$, the GNU becomes an auto-associator [Ale90c]. Additional parameters of the GNU network include the width W of the external input terminal and the degree of generalisation G in the network GRAMs. A schematic diagram of a GNU is shown in Figure 2.10.

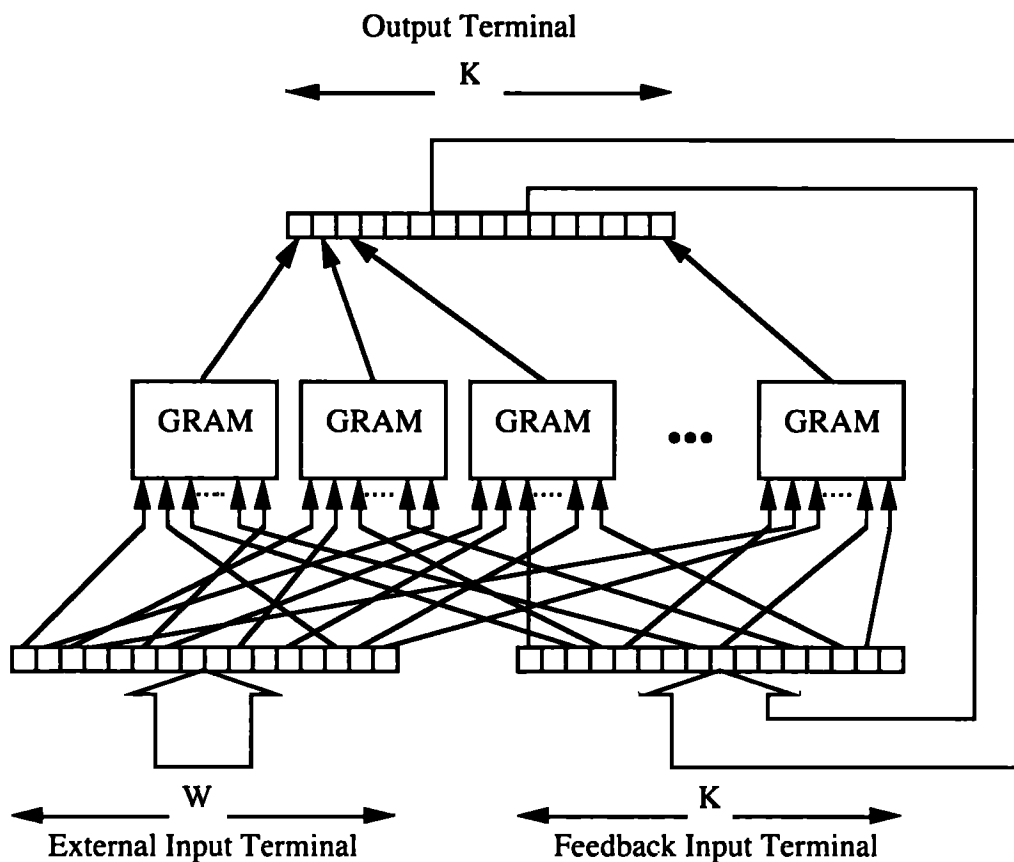


Fig. 2.10: The general neural unit network.

The functional properties of the GNU network are determined by the relative amount feed-back in the network. In [AleM91], the retrieval performance of the network is studied with respect to parameters N and F and theoretical predictions of optimal feedback for best retrieval performance are given.

The feed-back structure of the GNU network determines both its *associative* and *temporal* properties. The associative properties of the network result from the creation of stable states in the network, enabled by the presence of feed-back seen as being of the same type as in a Hopfield network. The temporal properties of the network result from the capacity of the network to learn state transitions, in which case the feed-back connections can be interpreted as coding the internal state of a Neural State Automaton (NSA). The *neural* characteristic of the resulting state transition diagram is mainly due to the fact that the transitions, from one state to another or from one state to itself, are those taught to the network during the learning phase. These transitions can be seen as involving prototype internal states of the network, or *macrostates*. To each macrostate, correspond many *microstates*, slightly different from the macrostate at the pixel level, but which behave macroscopically in a similar way, due to the generalisation ability of the network.

In Chapter 6, as a thesis contribution, the probability of disruption of patterns stored in a weakly interconnected GNU, used as an auto-associator, are calculated. It will be shown that the network can store about the same number of patterns as the number of inputs per node, in the case of equally and maximally distant patterns. A method is also suggested in order to increase the storage capacity of the network, by use of multiple GRAM nodes, with a structure similar to discriminator nodes. In Chapter 7, as a thesis contribution, the retrieval equations of the network will be established in the case of arbitrary stored patterns.

2.5.4. Self-organising weightless networks and unsupervised learning

A review of the work on self-organising neural networks, relevant to this thesis (Kohonen network, Allinson network and other models) is carried out in Chapter 5, together with the development of the C-discriminator network.

2.6. Summary

This chapter has first reviewed the weighted-sum-and-threshold neural model. This node model is often trained by gradient descent methods derived from the Hebbian learning paradigm. This was followed by a review of a series of weightless node models, for each of which the structure, the learning and generalisation properties were analysed.

The basic component of WANNs is the RAM which can be viewed as a bit-organised random access memory, as a look-up table or as a universal logic element. The RAM node is able to learn any logical function of its inputs by recording the function value for each possible combination of the input values. This is the first property characterising weightless models and it has been referred to as node loading. However, the basic RAM has no generalisation ability. In the derived weightless node models, three properties are added to the RAM model to endow it with generalisation properties. These properties have been

referred to as node decomposition, probabilistic contents and spreading.

Two types of node decomposition have been defined: the discriminator decomposition and the pyramidal decomposition. One of the parameters characterising both structures is the N -tuple size or number N of inputs to each sub-node of the structures. In addition, the discriminator node is characterised by K , the number of RAMs in the discriminator, and the pyramid node is characterised by D , the depth or number of layers in the pyramid. The parameters K and N for the discriminator, and D and N for the pyramid allow a measure of control over the generalisation and discrimination capabilities of both node models. Node decomposition has also the additional advantage of reducing the amount of memory required for the implementation of the nodes.

The introduction of a probabilistic contents u and, more generally, of multi-valued contents in the RAM node, has led to the definition of the PLN and its generalisations, the MPLN and C-RAM. There are four main reasons for the introduction of an undefined state u : it implements a "don't know" state; it enables the initialisation of the node memory contents to an un-biased state; in recurrent networks, the possibility of cyclic dynamics is reduced; and it enables the training of hidden nodes in multi-layer nodes (pyramids) or networks.

Another way of providing the RAM with generalisation capabilities is by using a spreading algorithm, performed during or after training, to spread the information contained in trained memory locations to untrained neighbouring locations. Spreading can be performed in a separate phase, after the learning phase, as in the GRAM model. Spreading can also be performed after each training step, as in the C-RAM and CDN models, which will be described in Chapter 5. When spreading is performed dynamically, as training proceeds, this leads to the definition of the DGRAM model, which will be described in Chapter 8. The C-RAM, CDN and DGRAM models are contributions to this thesis.

The structure of WANNs can be described by a number of levels which characterise whether there is feed-back in the network or not, and if so, what form it takes. Level-0 structures are feed-forward networks. In level-1 structures, feed-back is present and implements a mechanism of short term memory of the last network response. Level-2 structures are able to store pattern prototypes as inner states, and they exhibit a number of associative and temporal network behaviours. In level-3 structures, a second level of feed-back is added in order to implement an attention mechanism in the network.

Two important feed-forward weightless networks have been reviewed: the single layer discriminator network, which was used as a classifier; and the advanced distributed associative memory network, a two-layer network without hidden units, which was used as an hetero-associator.

Several studies of recurrent WANNs have been reviewed. These were concerned, on one hand, with recurrent PLN networks, sparsely connected (in Section 2.5.3.1) and fully-

connected (in Section 2.5.3.2.1), and on the other, with recurrent GRAM networks, fully-connected (in Section 2.5.3.2.2) and sparsely connected (in Section 2.5.3.3).

CHAPTER III. THE GENERALITY OF THE WEIGHTLESS APPROACH

3.1. Introduction

In contrast with Chapter 2 in which specific weightless models were reviewed, in this chapter a wider perspective is adopted and the generality of the weightless or RAM-based approach is argued from several complementary points of view.

The logic formalism inherent to WANN models is first discussed and it is shown how the neural activities of both McCP and RAM nodes can be expressed as logic functions. It is also shown that this logical formalism leads to a straightforward RAM-based implementation of McCP networks. The generality of weightless models with respect to the choice of node function set is then discussed in the light of results on the complexity of learning in neural networks. The next section reviews several pattern recognition methods and shows how these relate to weighted and weightless models. This is followed by a succinct review of weightless implementations of major neural learning paradigms. WANNs are then shown to belong to the class of emergent property systems. Finally, weighted and weightless systems are compared with respect to their connectivity, functionality, implementation, learning, generalisation, and distributed and localised representation properties.

3.2. Generality with respect to the logical formalism

3.2.1. Neuronal activities of McCP and RAM neurons

In this Section, the generality of the weightless approach is demonstrated by showing that the logical formalism used to express the function performed by weightless nodes is very general and is equally applicable to McCP neurons. Two expressions are derived for the neuronal activities of McCP and RAM neurons. Usually, the two types of neuronal models are compared by expressing the activation laws in the form of various algebraic expansions involving continuous variables (e.g., [Cul71], [Cai89], [Gur89], [Mar89] or [ClaGT89]). Here, the neuronal activities of both models are expressed as logic functions.

For the McCP, the same development as in [McCulP43] is followed. The Boolean variable y_i , called *action* of neuron c_i , takes the value 1 when c_i fires and 0 when it is quiescent. The number of excitatory synaptic connections from c_i to c_j is denoted w_{ji} . In the example of Figure 1.2 (see Section 1.2.3), these numbers are: $w_{51} = 2$, $w_{32} = 1$ and

$w_{54} = 3$. E_j and I_j denote the sets of indices attached to the names of the neurons that have excitatory or inhibitory synaptic connections upon c_j , respectively. In the example of Figure 1.2, $E_5 = \{1, 2, 4\}$ and $I_5 = \{3\}$. If θ_j is used to denote some threshold value associated with neuron c_j , then the set K_j is defined as the set of all subsets L_k of E_j which contain a number of elements exceeding θ_j . It holds that

$$K_j = \{L_k \mid L_k \subset E_j : \sum_{i \in L_k} w_{ji} > \theta_j\} \quad (3.1)$$

with $\sum$ being the algebraic sum. In the example of Figure 1.2, if a threshold $\theta_5 = 2$ is chosen, then:

$$K_5 = \{\{4\}, \{1, 2\}, \{1, 4\}, \{2, 4\}, \{1, 2, 4\}\}.$$

The action y_5 of neuron c_5 can then be expressed as a Boolean function of the actions of the neurons having synaptic connections upon it:

$$y_5 = \bar{y}_3(y_4 + y_1y_2 + y_1y_4 + y_2y_4 + y_1y_2y_4) = \bar{y}_3(y_4 + y_1y_2) \quad (3.2)$$

in which the signs + and - denote the OR and NOT connectives, respectively. A general expression for the law of nervous activation, as defined in [McCulP43], can therefore be expressed as:

$$y_i = \prod_{m \in I_i} \bar{y}_m \cdot \left\{ \sum_{L_k \in K_i} \prod_{s \in L_k} y_s \right\} \quad (3.3)$$

in which the $\sum$ and $\prod$ symbols denote the logical sum and product, respectively. It is interesting to note that the factor under curly braces in (3.3) is written as a sum of implicant¹ terms. Because of (3.1), these implicant terms can be replaced by the corresponding minterms in a straightforward fashion and (3.3) becomes:

$$y_i = \prod_{m \in I_i} \bar{y}_m \cdot \left\{ \sum_{L_k \in K_i} \left(\prod_{s \in L_k} y_s \cdot \prod_{t \in (E_i \setminus L_k)} \bar{y}_t \right) \right\} \quad (3.4)$$

¹ In the theory of logic circuit design [Flo64], an *implicant* of a logical function $f(x_1, x_2, \dots, x_n)$ is either a minterm of that function or some product term made up from a combination of suitable minterms of the function by using the reduction formula [Gre86];

$$(x_n x_{n-1} \dots x_{k+1} x_k x_{k-1} \dots x_1) + (x_n x_{n-1} \dots x_{k-1} \bar{x}_k x_{k-1} \dots x_1) = x_n x_{n-1} \dots x_{k+1} x_{k-1} \dots x_1.$$

The law of nervous activation of a McCP neuron can now be expressed in a canonical form, as a sum of minterms:

$$y_j = \sum_{L_k \in K_j} \prod_{\substack{s \in L_k \\ i \in (E_i \setminus L_k) \\ m \in I_k}} y_s \bar{y}_i \bar{y}_m. \quad (3.5)$$

For a RAM neuron, a similar but more general expression exists. A_j denotes the set of indices of the neurons having synaptic connections upon c_j :

$$A_j = \{i_1, i_2, \dots, i_N\}.$$

K_j is defined as the set of *all* subsets L_k of A_j :

$$K_j = \{L_k \mid L_k \subset A_j\}.$$

Choosing a subset M_j of K_j ($M_j \subset K_j$), the activation law of a RAM neuron receiving inputs from N other neurons, can be expressed as:

$$y_j = \sum_{L_k \in M_j} \prod_{\substack{s \in L_k \\ i \in (A_j \setminus L_k)}} y_s \bar{y}_i. \quad (3.6)$$

Neuron c_j performs a logical function of its inputs. Which function is performed depends on which minterms are present in (3.6), which in turn depends on which subset M_j is considered. There are 2^{2^N} such subsets possible.

The previous expressions show that, despite the fact that McCP neurons can be expressed in a way which is most natural for RAM neurons, the functionalities of the two node types are different. RAM neurons have a higher functionality.

3.2.2. RAM implementation of McCP networks

The following example shows how it is possible to implement networks of McCP neurons as networks of RAM nodes. The same example is used as in [McCulP43], the case of heat transient produced by a transient cooling:

*If a cold object is held to the skin for a moment and removed, a sensation of heat will be felt; if it is applied for a longer time, the sensation will be only of cold, with no preliminary warmth, however transient. It is known that one cutaneous receptor is affected by heat, and another by cold" [McCulP43].

The Boolean variables S_h and S_c are chosen to represent the actions of the cutaneous receptors to the heat stimulus and cold stimulus, respectively. Likewise the Boolean variables R_h and R_c are chosen to represent the actions of the response neurons whose activity implies a sensation of heat and cold, respectively. The above requirements can be written by the following logical functions of Boolean variables (with temporal dependence which is finite in the past [McCulP43]):

$$\begin{cases} R_h(t) = S_h(t-1) + S_c(t-3) \cdot \bar{S}_c(t-2) \\ R_c(t) = S_c(t-2) \cdot S_c(t-1). \end{cases} \quad (3.7)$$

The system of equations (3.7) can be rewritten using a temporal operator T , defined as

$$X(t - \tau) = T^\tau(X).$$

Equations (3.7) become therefore

$$\begin{cases} R_h = TS_h + T^3S_c \cdot T^2\bar{S}_c = T(S_h + T(TS_c \cdot \bar{S}_c)) \\ R_c = T^2S_c \cdot TS_c = T(TS_c \cdot S_c). \end{cases} \quad (3.8)$$

The factorisation in (3.8) provides a means of constructing the network, using 2 additional neurons whose actions I_a and I_h define the internal state of the system:

$$\begin{cases} R_h = T(S_h + I_h) \\ R_c = T(I_a \cdot S_c) \\ I_a = T(S_c) \\ I_h = T(I_a \cdot \bar{S}_c) \end{cases} \quad (3.9)$$

The system of logical equations (3.9) is analogous to a system of first-order differential equations encountered in 'analogue' models. The temporal operator in (3.9) can be eliminated and it holds:

$$\begin{cases} R_h = s_h + i_h \\ R_c = i_a \cdot s_c \\ I_a = s_c \\ I_h = i_a \cdot \bar{s}_c \end{cases} \quad (3.10)$$

in which a Boolean variable x represents the actual activation of a neuron at time t , whereas the Boolean variable X , associated with x , represents the value of x at the next

instant in time, in the case of a synchronous logical system² [Flo64]. Figure 3.1 shows two representations of the system described by (3.10), as a network of McCP neurons (a) and implemented as a synchronous sequential logic circuit (b).

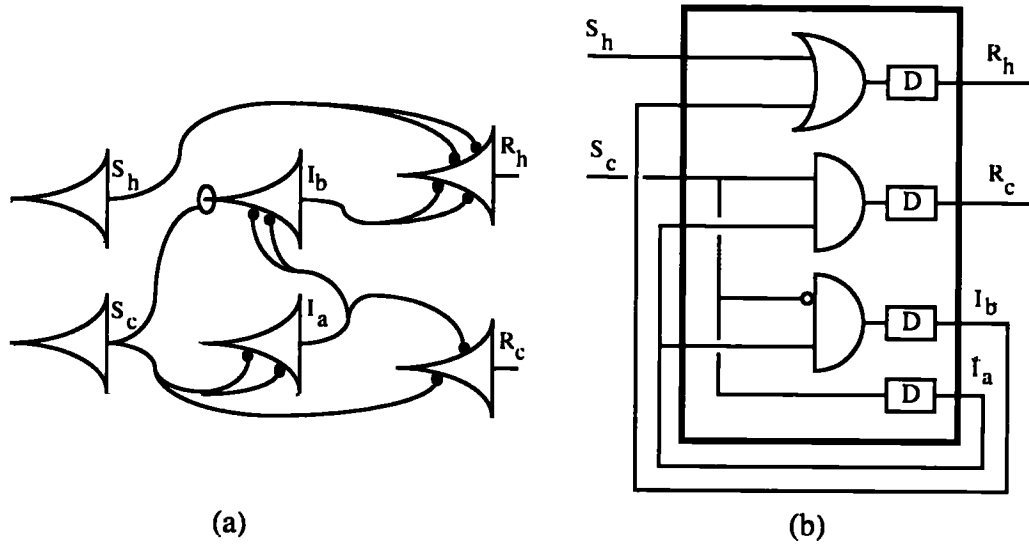


Fig. 3.1: (a) A network of McCP neurons and (b) its implementation as a synchronous sequential logic circuit.

Figure 3.1 (b) shows that the system has feed-back, despite the fact that the McCP network itself can be represented as a network without circles (Figure 3.1 (a)). This suggests the usefulness of an automaton-based approach, as already mentioned in Section 1.2.5.

In the previous paragraphs, learning did not occur. The structure of the network was derived from the desired functionality. The WST interpretation of the McCP neurons provides a learning mechanism for the system. Similarly, the RAM interpretation provides a learning mechanism for the finite state machine structure of Figure 3.1 (b). The approach is the same in both cases: internal adaptive variables are assumed which will enable learning to take place. The weighted and weightless implementations of a learning system are shown in Figure 3.2.

² The theory was generalised to cover the case of asynchronous logical systems. In these systems, to each variable x are associated a switch-on delay ϵ_x and a switch-off delay δ_x . If $x = 0$ and $X = 1$, the value of x will switch to $x = 1$ after a delay ϵ_x , unless X returns to 0 in the meantime, in which case the command for x to switch to 1 is cancelled. Likewise, if $x = 1$ and $X = 0$, the value of x will switch to $x = 0$ after a delay δ_x , unless X returns to 1 in the meantime, in which case the command for x to switch to 0 is cancelled. This formalism has been referred to as Kinetic Logic [V-Ham75][V-Ham84]. It enables the study of logical dynamical systems which exhibit very complicated state transitions, as a result of the complex race conditions arising between the different variables involved. Kinetic Logic was successfully applied to the modelling of chemical and genetic systems, using a logic approach [Tho79].

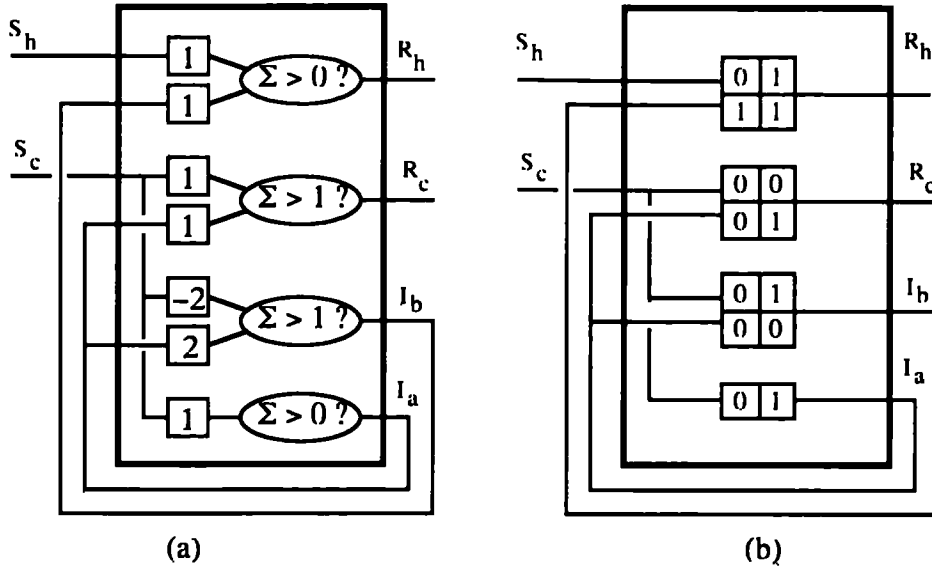


Fig. 3.2: (a) Weighted and (b) weightless implementations of a learning mechanism for a network of McCCP neurons.

3.3. Generality with respect to the node function set

Weighted-sum-and-threshold functions (WST) have become standard node functions in neural network studies, but the reasons for using them are not well founded. In [Jud90], the rationales for, and trade-offs among various node function sets are examined, in the context of the complexity of learning. The question arises of whether there are other types of functions that might be more justifiable, or might work better, or might make learning more tractable.

In [Jud90], a neural network is defined loosely as made of computing nodes, communication links and different message types. The network architecture as well as the task to be performed by it, denoted A and T respectively, are seen as variables. *Loading* is then the process of assigning an appropriate function to every node in the architecture so that the resulting function f learned by the network includes the set of input-output associations defining the task. Thus,

$$f = \text{loading}(A, T). \quad (3.11)$$

The loading problem amounts to a search problem. It is associated with the *functionality problem*, which is to determine whether a given network architecture can perform the imposed task. If it cannot, then a procedure should announce that fact. This relates to the questions of functional capacity, reviewed in Section 2.3.5.3 for pyramidal structures, and also to the choice of appropriate node function set.

The loading problem is used to make two important points. On one hand, the learning or memorisation problem in its general form is intractable. The loading problem is NP-complete:

“there is no reliable method to configure a given arbitrary network to remember a given arbitrary body of data in a reasonable amount of time” [Jud90].

The intractability of memorisation implies the intractability of generalisation. On the other hand:

“there are many ways to circumvent this negative result, and each one corresponds to a particular constraint on the learning problem. There are fast learning algorithms for cases where the network is of a very restricted design, or where the data to be loaded are very simple” [Jud90].

In [Jud90], the question is asked whether the type of node functions typically used in the connectionist literature is justified and appropriate. It is argued that the difficulty of the loading problem is independent of the choice of function set that each node can perform. The complexity of loading is not a consequence of the node function set but, rather, of the connectivity patterns of the network.

Where the node functionality matters, however, is in the case of single layer networks, for which the node function set has an overwhelming effect on what can be performed by the network.

In conclusion, from the point of view of the generality of the weightless approach, the results obtained by Judd on the complexity of learning in neural networks have three consequences. First, it is confirmed that, in general, the learning capabilities of the network are mostly dependent on its topological structure and not on the choice of node function set. Second, the first conclusion allows for the choice of any node function set. Therefore, the choice of the RAM-based neurons is as good as any other function set and has various advantages (such as, for example, its easy implementation). Third, in the case of single-layer networks, the node function set is important. Therefore, choosing RAM-based neuron models guarantees the largest possible functionality. The case of single-layer networks is also of interest for networks with feed-back connections, such as the general neural unit, exposed in Section 2.5.3.3.

3.4. Generality with respect to pattern recognition methods

3.4.1. Introduction

In this Section, the generality of the discriminator network, described in Section 2.5.2.2, is demonstrated by showing how its operation, based on the Bledsoe and Browning N-tuple method (BBNM), relates to the Maximum Likelihood Method (MLM) and to the Maximum

Likelihood N-tuple Method (MLNM). It is also shown that the function performed by the discriminator network is a degenerate form of the Nearest Neighbour N-tuple Method (NNNM). The particular case of the Nearest Neighbour Hamming Distance Method (NNHDM) is also examined. Most of the following analysis is due to Ullmann [Ull73].

3.4.2. The maximum likelihood decision rule

A classification problem with z recognition classes is considered: $\{C_1, C_2, \dots, C_i, \dots, C_z\}$. Its implementation as a single layer network of z neurons, with each recognition class being assigned to a different neuron, is assumed.

The following notations are used: $P(r)$ denotes the *a priori* probability (before the pattern is actually presented to the network) that an input pattern belongs to class C_r ; $P(X)$ denotes the probability that an input pattern is the pattern X ; $P(X/r)$ denotes the probability that an input pattern belonging to C_r is pattern X and $P(r/X)$ the *a posteriori* probability (after the pattern has been presented to the network) that input pattern X belongs to C_r .

An unknown pattern X is assigned to class C_r such that the *a posteriori* probability of X belonging to class C_r is higher than the *a posteriori* probability of X belonging to any other class C_s . Mathematically, this is expressed as:

$$X \in C_r : \forall r \neq s : P(s/X) > P(r/X). \quad (3.12)$$

This rule minimises the probability of misrecognition of X . Equation (3.12) implies that $P(r/X)$ needs to be estimated. In fact, it is $P(X/r)$ which is first estimated, by making suitable assumptions concerning the statistics of the recognition classes, and then, using Bayes rule, $P(r/X)$ is derived:

$$P(r/X) = \frac{P(r) \cdot P(X/r)}{P(X)}. \quad (3.13)$$

Substituting (3.13) into (3.12), the Maximum Likelihood Decision Rule (MDLR) is obtained:

$$X \in C_r : \forall r \neq s : P(s) \cdot P(X/s) > P(r) \cdot P(X/r). \quad (3.14)$$

3.4.3. The maximum likelihood method

The Maximum Likelihood Method of classification (MLM) is based on the assumption of statistical independence between input pixels in the input patterns.

A binary pattern X is represented as a vector $\bar{x} = (x_1, x_2, \dots, x_j, \dots, x_m)^T$ and n_{rj} denotes the number of patterns belonging to class C_r for which $x_j = 1$. Then, if t_r is used to denote the size of the training set for class C_r , the following estimates are obtained:

$$P((x_j = 1) / r) \approx p_{rj} = \frac{n_{rj}}{t_r}$$

and

$$P((x_j = 0) / r) \approx 1 - p_{rj}.$$

From the assumption of independence between input pixels, it follows that:

$$\begin{aligned} P(X / r) &= P(x_1 / r) \cdot P(x_2 / r) \cdots P(x_m / r) \\ &= \prod_{j=1}^m p_{rj}^{x_j} \cdot (1 - p_{rj})^{1-x_j} \end{aligned} \quad (3.15)$$

Taking the logarithm in each side of (3.15) yields

$$y_r = \sum_{j=1}^m x_j \cdot w_{rj} - \theta_r, \quad (3.16)$$

with

$$y_r = \log P(X / r), \quad (3.17)$$

$$w_{rj} = \log \frac{p_{rj}}{1 - p_{rj}}$$

and

$$\theta_r = - \sum_{j=1}^m \log(1 - p_{rj}).$$

It is worth noting the formal equivalence of (3.16) with the output function (2.4) of a linear neuron (LU). Making the additional assumption that the a priori probabilities $P(r)$ are the same for all classes, the MLDR becomes that X is assigned to the class for which $P(X / r)$ is the greatest or, equivalently, for which y_r is the greatest:

$$X \in C_s : \forall r \neq s : y_i > y_{r,i} \quad (3.18)$$

It is clear that the decision process described by (3.18) could be implemented as a single layer of LUs with output function described by (2.4).

3.4.4. The maximum likelihood N -tuple method

The Maximum Likelihood N -tuple Method (MLNM) is based on the assumption of statistical independence between N -tuples of input pixels (the same assumption was made in Section 2.5.2.2.3), an N -tuple being defined as a ordered set of N pixels $\{x_1, x_2, \dots, x_N\}$. A arbitrary number K of N -tuples is chosen. The state S of an N -tuple is defined by the pixel values of the N -tuple.

A particular state S of a particular N -tuple is considered. If n_{rS} is used to denote the number of patterns of class C_r which include the state S , then the proportion $\frac{n_{rS}}{t_r}$ of patterns of class C_r for which the considered N -tuple is in state S can be used as an approximation for $P(S/r)$.

The elements of the i -th of the K N -tuples are denoted $\bar{x}_i = \{x_1, x_2, \dots, x_N\}$. There are 2^N possible N -element binary patterns: $Z_1, Z_2, \dots, Z_j, \dots, Z_{2^N}$. Pattern Z_j has elements $\{z_{j1}, z_{j2}, \dots, z_{jN}\}$. For each of these Z_j and for each N -tuple $\bar{x}_i$, a function $\phi_{ij}(X)$ is defined:

$$\phi_{ij}(X) = x_1^{z_{j1}} \cdot (1 - x_1)^{1-z_{j1}} \cdot x_2^{z_{j2}} \cdot (1 - x_2)^{1-z_{j2}} \cdot \dots \cdot x_N^{z_{jN}} \cdot (1 - x_N)^{1-z_{jN}}.$$

For example, for the i -th N -tuple (with $N = 3$):

$$\begin{cases} \phi_{i0} = (1 - x_1) \cdot (1 - x_2) \cdot (1 - x_3) \\ \phi_{i2} = (1 - x_1) \cdot (1 - x_2) \cdot x_3 \\ \vdots \\ \phi_{i7} = x_1 \cdot x_2 \cdot x_3 \end{cases}$$

Making the assumption of statistical independence between N -tuples, $P(X/r)$ can be approximated as the product of the N -tuple state conditional probabilities for all N -tuples:

$$P(X/r) = \prod_{i=1}^K \sum_{j=1}^{2^N} \mu_{r,ij} \cdot \phi_{ij}(X) \quad (3.19)$$

with

$$\mu_{r,ij} = \sum_X P(X/r) \cdot \phi_{ij}(X) \approx \frac{1}{t_r} \sum_{k=1}^{t_r} \phi_{ij}(X_{rk}) \quad (3.20)$$

In other words, $\mu_{r,ij}$ is approximated as the average of $\phi_{ij}(X)$ over the training set of class C_r . If the RAM paradigm is used, $\mu_{r,ij}$ can also be interpreted as the proportion of times that the memory location j of the i -th RAM has been addressed by the t_r patterns of class C_r . The sum in (3.19) can also be interpreted as the decoding operation taking place when the RAMs are addressed by an input pattern. The aim of the decoding operation is to choose the particular $\mu_{r,ij}$ corresponding to the location addressed by X in RAM i . Indeed, for a given X , $\phi_{ij}(X)$ is equal to 1 for one and only one value of j in the range $[1, 2^N]$, and for all other values of j , $\phi_{ij}(X)$ is equal to 0. Taking the logarithms in both sides of (3.19) yields

$$y_r = \sum_{i=1}^K \sum_{j=1}^{2^N} \phi_{ij}(X) \cdot m_{r,ij}, \quad (3.21)$$

with (3.17) and with

$$m_{r,ij} = \log \mu_{r,ij}.$$

Finally, using (3.21) instead of (3.16), the MLDR takes the same form as (3.18).

3.4.5. The nearest neighbour N -tuple method

The Nearest Neighbour N -tuple Method (NNNM) consists of storing the patterns of the training set of every recognition class, in the recognition machine. Each pattern is stored as a set K N -tuples.

If X_{rk} denotes the k -th training pattern for class C_r and M represents the set of K N -tuples of an unknown pattern X , the subset M_{rk} of M ($M_{rk} \subset M$) is defined as the set of N -tuples from M which have the same state in X_{rk} . The MLDR (3.18) can then be used, with

$$y_r = \max_{k=1}^{t_r} \|M_{rk}\|, \quad (3.22)$$

in which the values $\|M_{rk}\|$, denoting the sizes of the sets M_{rk} , must be calculated separately for each training pattern. In the particular case corresponding to $N = 1$ and $K = m$, the method is called Hamming distance Neighbour Method (NNHDM).

3.4.6. The discriminator network

The learning phase in a discriminator network, as described in Sections 2.3.2, 2.3.3 and 2.5.2, consists, for a certain discriminator, of storing a value 1 at the memory locations addressed by each N -tuple of each training pattern. Therefore, what is recorded is not, as in the MLNM, the number of occurrences of each state of each N -tuple in the training set of each class, but simply which states of each N -tuple occur *at least once* in the training set of each class. The method is also referred to as the Bledsoe and Browning N -tuple method (BBNM) [Ull73].

In the BBNM, the MLDR (3.18) is still applicable, but with

$$y_r = \sum_{j=1}^K \sum_{i=1}^{2^N} \phi_{ij}(X) \cdot u_{r,ij} \quad (3.23)$$

using the step function

$$u_{r,ij} = \begin{cases} 1 & \mu_{r,ij} > 0 \\ 0 & \mu_{r,ij} = 0. \end{cases}$$

In this case, (3.18) corresponds to the function of the decision circuit of Section 2.5.2.2.1, represented in Figure 2.8, which assigned the classification of an unknown pattern to that discriminator node which had the strongest response. Equation (3.23) can be interpreted as the output function performed by a discriminator node.

It is interesting to compare the influence of the N -tuple size in MLNM and in the discriminator network. In MLNM, as N increases, say from 2 to 10, a better estimate of y_r is made, by taking more account of the statistical dependence between pattern elements. But, as N becomes large, say $N > 10$, the N -tuple states have ever lower probability of occurrence, and therefore these probabilities are less accurately estimated by (3.20). This affects therefore the recognition performance negatively. To be effective, the MLNM needs a larger training set for larger values of N . This is not the case for the discriminator network, whose performance does not depend on a probability estimation like (3.20), whose accuracy depends on the number of training patterns. Furthermore, experimental results have shown that the discriminator network reaches a higher overall probability of correct recognition (over all values of N) than does the MLNM [Ull73].

The operation of a discriminator network can also be viewed as a degenerate form of the NNNM. Indeed, using the same notations as in Section 3.4.5, M_r can be defined as the set of N -tuples which have the same state in *at least* one of the X_n :

$$M_r = \bigcup_{i=1}^{I_r} M_{ri}.$$

Hence, the MLDR (3.18) is used with

$$y_r = \|M_r\|, \quad (3.24)$$

which is equivalent to (3.21). Clearly, this shows that the nearest neighbour method used by the discriminator network is a degenerate one, since it takes no account of joint occurrences of N -tuple states in the same training pattern. Its performance is expected to approximate the performance of the non-degenerate method (NNNM) more closely as N gets bigger [Ull73].

Finally, it is worth noting the use of *polynomial* discriminant functions in the MLNM, the discriminator network and in weightless nodes in general, whereas *linear* discriminant functions are used in the MLM and with McCP nodes in general [Ull73] [MinP69].

3.5. Generality with respect to standard neural learning paradigms

Many of the standard neural network learning paradigms, usually implemented with neuron models derived from the McCP neuron, have also been implemented using weightless neurons. In this section, references are given for the most important of these weightless implementations.

The Error Back Propagation (EBP) algorithm or GDR, reviewed in Section 2.2.2, has been implemented by many researchers (e.g., [RumHW86]) in networks of WST neurons, most often referred to as Multi-layer Perceptron (MLP) networks. Several weightless implementations of the paradigm have also been carried out. In [TatFL89], a network of discriminators, referred to as Single-layer Look-up Perceptrons (SLLUP), is trained by gradient descent and shown to converge faster than a MLP network on the fuzzy XOR problem. In [Mar89], EBP is applied to pyramidal PLN networks and in [Kru91], to networks of LIBN nodes. In [Gur89], a proof is given for the convergence of a similar algorithm under stochastic gradient descent.

Another popular neural learning paradigm is Kohonen's Topologically-Ordered Feature Map (TOFM) (e.g., [Koh89a]). Weightless implementations of this paradigm include [AllJ89], [Nto90] and [TamS92]. Chapter 5 is devoted to this type of unsupervised learning and the C-discriminator network is introduced as a thesis contribution.

Weightless networks with a same configuration as in a Hopfield network [Hop82] are studied in [Ale89a], [WonS89] and [Nto91], and simulated annealing methods similar to

those used in Boltzmann machines [AckHS85] are discussed, in a 'weightless' context, in [Ale87], and implemented in [Ale89a], in the form of a noise training algorithm. In [CarP87a] (also in [CarP87b]) and [VdBroK90], the simulated annealing technique is applied to the optimisation of the input connection mapping of a feed-forward RAM network.

For completion, weightless implementations of learning schemes similar to Grossberg's Adaptive Resonance Theory (ART) paradigm (e.g., [CarG89]) can be found, for example, in [CarFBF91], [KerS90] or [Ful93]. The implementation of reinforcement learning methods [BarSA83] using weightless neuron models can be found, for instance, in [Mye90], [GorT90], [Gur89] or [PenGS91]. Finally, recurrent networks [Jor88] have also been studied with weightless networks in [AleH76], [Nto91] or [AleEP93].

3.6. Generality with respect to emergent property systems

In the same way as there is a cleavage between weighted and weightless neural systems at the node level, a cleavage exists between Von Neumann architectures and neural architectures at the system level [Ale87]. Both weighted and weightless neural systems belong to the category of systems endowed with *emergent* properties, as opposed to *algorithm-driven* systems.

In algorithm-driven systems, the representation of information is localised in different parts of the memory. In neural networks, the knowledge is distributed throughout the system. It is the collaboration between the neural elements that enable the system to retrieve stored information or to make decisions upon the presentation of input patterns to the system. Systems whose properties are determined by the collaboration of numerous elementary components are referred to as systems with emergent properties. In these systems, sophisticated overall network behaviour derives from unsophisticated components [Jud90]. Learning itself is seen as the capacity of the system to absorb information from its environment without requiring some external agent to program it. The result is a system that can store and retrieve knowledge in a way that cannot be accomplished by algorithmic methods except through inefficient, exhaustive searches [Ale87].

3.7. Weightless versus weighted neural systems

3.7.1. Connectivity versus functionality

When comparing weightless neural networks with McCP-type neural networks, a duality can be formulated between the node functionality and network connectivity of both models.

On one hand, weightless nodes, such as the RAM, PLN or GRAM, implement a universal node function set, and decomposed structures such as the discriminator or

pyramid are made of sub-nodes which also have a complete functionality. This is to be compared with the McCP neurons which only implement the linearly separable function set. The weightless decomposed structures still have a much higher functionality than WST neurons with equivalent number of inputs.

On the other hand, a lower nodal connectivity is necessary in RAM-based networks than in McCP networks. Weightless networks, such as discriminator-based or GNU networks, do not need to be fully connected, as in a Hopfield networks [Hop82] or have full connectivity between layers, as in a MLP network [RumHW86].

3.7.2. Ease of implementation

One of the most obvious advantages of weightless systems over traditional WST neural modes is their propensity for immediate implementation as electronic RAM circuits. Machines such as those presented in Section 1.2.6 are examples of such hardware implementations. In such machines, the RAM is seen as the simplest possible learning processor. The addressing operations in RAMs also offer a directness for making fast machines [Ale83a].

However, the hardware implementation of more elaborate weightless models, such as the GRAM or DGRAM, is not at all as straightforward as is the implementation of, say, the simple discriminator node. Indeed, for weightless nodes whose operation involves pseudo-random number generation, updating of multi-valued memory contents or spreading of information to neighbouring memory locations, the actual hardware implementation requires an amount of circuitry comparable to that necessary for the implementation of WST models.

In conclusion, the ease of implementation of weightless systems is only relevant for some of the models and it may well be that a software implementation is more appropriate for other models (e.g., [AleEP93]). So, implementation issues are not central to the weightless approach

“because embodiments of resulting neural networks might turn out to be software for conventional processors, specialised microprocessor systems or straightforward read-only memory programmable logic array hardware” [AleS79].

What is central, however, is the ability of the weightless networks to learn, discriminate and generalise.

3.7.3. Learning and generalisation

The learning and generalisation properties of weightless neural networks has been reviewed in detail in Chapter 2, the conclusion being that these systems have the ability to learn with

controlled functionality and generalisation.

At the node level, the generalisation properties of weightless systems are implemented either through training with noise, node decomposition or spreading procedures.

At the network level, the generalisation properties of weightless systems are implemented through node connectivity, with or without feed-back between nodes.

Efficient learning schemes for weightless neural networks have been shown to exist which involve techniques such as noise training, global reward-punishment and spreading.

In general, weightless neural systems learn faster than equivalent weighted systems. This is due to the inherent speed of the read and write memory operations. For decomposed pyramidal structures learning is only faster for small networks. Issues concerning learning complexity have already been discussed in Section 3.3.

In general, weightless neural systems have a higher storage capacity than equivalent weighted systems. Questions of functional capacities and storage capacities have already been discussed in detail in Chapter 2 and further results are obtained, as thesis contributions, in Chapters 4, 6 and 7.

3.7.4. Distributed and localised representations

Equations such (2.8), describing the internal representation of patterns in neurons with decomposed topologies, clearly show that weightless neural networks belong to the class of systems that represent information internally in a distributed manner.

However, because of the RAM structure of the nodes, there is a much greater *localisation* of the memory elements than in weighted nodes. When needed, however, appropriate *delocalisation* of memory representation can be achieved either by node decomposition or by spreading techniques. In [WonS89], it is argued that the localisation of information characterising weightless systems provides high efficiency for some hard learning problems for which localised memorisation is the most natural way to learn the task. For instance, in the parity checking problem, opposite binary values must be stored in neighbouring memory locations in each node.

3.8. Conclusions

In this chapter, the generality of the weightless approach has been argued. Several points of view were adopted.

The logical formalism, inherent to WANN models, was shown to be very general and to be applicable to the expression of the functions computed by both McCP and RAM neuron models. Also, this formalism leads to a direct implementation of McCP networks as RAM networks. It was also shown that the logic functions of McCP and RAM models were not identical and that the RAM has a larger function set than the McCP neuron.

The question of node function set was then discussed in respect of learning complexity results and the following conclusions were drawn. In general, the learning capabilities of a neural network are mostly dependent on its topological structure and not on the choice of the node function set. Therefore, any function set is equally valid and in particular, that of RAM neurons. However, in the case of single-layer networks, the node function set is important. Therefore, choosing RAM-based neuron models guarantees the largest possible functionality.

WANNs are based on the use of polynomial discriminant functions, which are more general than linear discriminant functions, used with McCP networks. Several pattern recognition methods, based on the maximum likelihood decision rule, have been reviewed and it was shown how these relate to single-layer weighted and weightless classification networks. Several advantages of weightless networks such as the single-layer discriminator network, have been demonstrated. First, the N -tuple sampling paradigm is based on the assumption of statistical independence between N -tuples of input pixels. This is a more general assumption than the assumption of independence between input pixels, corresponding to a single layer weighted-sum network. Second, the performance of the weightless network does not depend on probability estimations, whose accuracy depends on the number of training patterns available. And third, the performance of the system approximates that of a nearest-neighbour N -tuple classifier system.

The generality of WANNs has been demonstrated with respect to the implementation of standard neural network learning paradigms.

WANNs can store and retrieve knowledge as a result of a collaboration between many simple computational elements. They belong therefore to the class of emergent property systems.

Weightless models have a higher functionality than weighted models and require, in general, a lower nodal connectivity.

Some weightless models present the advantage of a straightforward hardware implementation, which is mainly due to their low connectivity requirement and their RAM-based structure. Other more sophisticated weightless models, involving pseudo-random number generation, multi-valued node contents and spreading procedures, present similar implementation challenges to those presented by weighted models.

WANNs have the ability to learn and their generalisation and discrimination properties can be controlled through appropriate parameter settings, both at the node and network levels.

Finally, the question of distributed and localised representations in WANNs has been discussed and it was shown that the degree of information distribution in these systems can be controlled by appropriate decomposition and spreading techniques.

CHAPTER IV. FURTHER PROPERTIES OF FEED-FORWARD PYRAMIDAL WEIGHTLESS NEURAL NETWORKS

4.1. Introduction

This chapter is concerned with feed-forward pyramidal PLN networks. This structure was reviewed in Section 2.3.5 as a decomposed weightless node model. In this chapter, results are obtained which augment what has been done by others in respect of the functionality, the storage capacity and the learning dynamics of the network.

An exact and non-recursive formula is derived for the functional capacity of a D -layer pyramidal network with 2-input nodes ($N = 2$), and it is shown that the functional capacity of the network grows as 6^{2D} . An approximation to the functional capacity of a pyramid with general parameters N and D is also derived.

The storage capacity of the network, defined as the number of arbitrary input-output associations that it can store, is investigated using an exhaustive search methodology. A resulting exact probability distribution is obtained for the case $N = D = 2$.

The ASA training algorithm, reviewed in Section 2.3.5.2, is considered and the learning dynamics of the pyramidal network, trained on the parity checking problem (PCP), are investigated. The number of solutions to the PCP is calculated, for the case of general network parameters N and D . A calculation of the transition probabilities of the pyramid's internal state, during training of the PCP, is performed for the case $N = 2$, $D = 2$; the contents of the pyramid is proved to converge towards a fully trained configuration.

4.2. Functionality of feed-forward pyramidal networks

4.2.1. Simple non-recursive formula

4.2.1.1. Introduction

The pyramidal network is defined by the number of layers, or *depth*, D , and by the number of inputs per node, or *connectivity*, N . In section 2.3.5.3, the notion of functional capacity of a regular pyramidal weightless node was introduced. Using the function $\phi(k)$ to represent the number of distinct Boolean functions that depend exactly on k *particular* variables, the functional capacity $\Gamma(N, D)$ was given by the recursive expressions

$$\Gamma(N, D) = \sum_{i=0}^N \binom{N}{i} \phi(N-i) \left[\frac{\Gamma(N, D-1) - 2}{2} \right]^{N-i} \quad (4.1)$$

and

$$\phi(k) = 2^{2^k} - \sum_{j=1}^k \binom{k}{j} \phi(k-j), \quad (4.2)$$

with the boundary conditions

$$\Gamma(N, 1) = 2^{2^N} \quad (4.3)$$

and

$$\phi(0) = 2. \quad (4.4)$$

Equations (4.1) and (4.2) are recursive in both the number of layers D of the network and the number of inputs N to each node. An *exact* and *non-recursive* formula is derived in the next sub-section, for the functional capacity of a D -layer network with 2-input nodes ($N = 2$), and it is shown that the functional capacity of the network grows as 6^{2^D} . It is not possible to derive non-recursive formulae for pyramids with parameter $N > 2$.

4.2.1.2. Derivation

The function $\Psi_{N,D}(k)$ is defined as the number of Boolean functions realisable by a pyramid with parameters N , D , and which depend on *exactly* k *non-particular* variables. Since the total number of inputs to the network is N^D , it holds that

$$\Gamma(N, D) = \sum_{i=0}^{N^D} \Psi_{N,D}(i). \quad (4.5)$$

It can be shown that, for $N = 2$ and only for that case, (4.5) can be written as

$$\Gamma(2, D) = \sum_{i=0}^{2^D} \binom{2^D}{i} \psi_D(i), \quad (4.6)$$

in which the function $\psi_D(i)$ is defined as the number of Boolean functions realisable by a

pyramidal network with parameters $N = 2$, D and which depend on *exactly k particular* variables. For example, the case $N = 2$, $D = 2$ is considered. Using (4.1), it holds that

$$\Gamma(2,2) = \sum_{j=0}^2 \binom{2}{j} \phi(2-j) \left[\frac{2^{2^j} - 2}{2} \right]^{2-j} = \phi(2) \cdot 7^2 + 2 \cdot \phi(1) \cdot 7 + \phi(0).$$

Using (4.2) and the boundary condition (4.4), it holds that

$$\begin{cases} \phi(0) = 2 \\ \phi(1) = 4 - \phi(0) = 2 \\ \phi(2) = 16 - 2\phi(1) - \phi(0) = 10. \end{cases}$$

Thus,

$$\Gamma(2,2) = 10 \cdot 7^2 + 2 \cdot 2 \cdot 7 + 2 = 520.$$

The values of the function $\psi_2(i)$ are now evaluated, so as to be able to use expression (4.6) in the calculation of $\Gamma(2,2)$. The calculation of $\psi_2(4)$ is first considered. In the top node, the functions to be taken into account are those depending on both input variables. Their number is $\phi(2)$. In each node of the bottom layer, there are $\frac{\phi(2)}{2}$ functions to take into account. Hence, using the notation

$$\phi'(k) = \frac{\phi(k)}{2},$$

the number $\psi_2(4)$ of functions which depend on all input variables can be expressed as

$$\psi_2(4) = \phi(2) \cdot \phi'(2) \cdot \phi'(2) = 250.$$

Similarly for the other values of $\psi_2(i)$, it holds:

$$\begin{aligned} \psi_2(3) &= \phi(2) \cdot \phi'(2) \cdot \phi'(1) = 50, \\ \psi_2(2) &= \phi(1) \cdot \phi'(2) \cdot \phi'(0) = \phi(2) \cdot \phi'(1) \cdot \phi'(1) = 10, \\ \psi_2(1) &= \phi(1) \cdot \phi'(1) \cdot \phi'(0) = 2, \\ \psi_2(0) &= \phi(0) \cdot \phi'(0) \cdot \phi'(0) = 2, \end{aligned}$$

and finally, using (4.6),

$$\Gamma(2,2) = \sum_{i=0}^4 \binom{4}{i} \psi_2(i) = 1 \cdot 2 + 4 \cdot 2 + 6 \cdot 10 + 4 \cdot 50 + 1 \cdot 250 = 520.$$

It is interesting to note that the $\phi(k)$ are universal parameters, whereas the $\psi_D(k)$ are dependent on the number of layers in the network. The $\psi_D(k)$ could therefore be used to compare the functional capabilities of different network configurations instead of using the overall measure $\Gamma(2,D)$ proposed in [A-Ala90]. Table 4.1 gives the values of $\psi_D(k)$, for the different values of k , in the cases $D=1$ and $D=2$.

	$\psi(0)$	$\psi(1)$	$\psi(2)$	$\psi(3)$	$\psi(4)$
$D = 1$	2	2	10	218	64,594
$D = 2$	2	2	10	50	250

Table 4.1: Values of $\psi_D(k)$, for the different values of k , in the cases $D=1$ and $D=2$.

The following lemma can now be stated, showing that the values $\psi_D(i)$ can be expressed as the elements of a geometrical series:

Lemma 4.1: *The number $\psi_D(k)$ of Boolean functions realisable by a pyramidal network, with parameters $N=2$ and any D , which depend on exactly k particular variables is given by the expression*

$$\psi_D(i) = \begin{cases} 2 & i=0 \\ 2 \cdot 5^{i-1} & i=1,2,\dots,2^D \end{cases} \quad (4.7)$$

Proof: A proof of (4.7) is given in Appendix A.1.

Theorem 4.1: *The functional capacity $\Gamma(2,D)$ of a D -layer weightless pyramidal network with 2-input nodes ($N=2$) is given by the expression*

$$\Gamma(2,D) = \frac{8}{5} + \frac{2}{5} \cdot 6^{2^D}. \quad (4.8)$$

Proof: The substitution of (4.7) into (4.6) yields

$$\Gamma(2,D) = 2 + 2 \cdot \sum_{i=1}^{2^D} \binom{2^D}{i} 5^{i-1}$$

$$\begin{aligned}
&= 2 + \frac{2}{5} \cdot \sum_{i=1}^{2^n} \binom{2^n}{i} 5^i \\
&= 2 + \frac{2}{5} \cdot [(5+1)^{2^n} - 1]
\end{aligned}$$

and (4.8) follows. This concludes the proof of theorem 4.1.

Alternative proof: Equation (4.8) can easily be verified by substitution into (4.1) and (4.2), with the value $N = 2$. This calculation is carried out in Appendix A.2.

4.2.2. Approximations

An approximation to the functional capacity of a pyramidal WANN has been calculated for the case $N \gg 1$.

Lemma 4.2: For $k \gg 1$, the number $\phi(k)$ of distinct Boolean functions depending on k particular variables can be approximated by the expression

$$\phi(k) \approx 2^{2^k} \quad (4.9)$$

Proof: A proof of (4.9) is given in Appendix A.3.

Theorem 4.2: For $N \gg 1$, the functional capacity of a pyramidal weightless neural network can be approximated the expression

$$\Gamma(N, D) \approx 2^{2^N \cdot N^{D-1}} \quad (4.10)$$

Proof: A proof of (4.10) is given in Appendix A.4.

Equation (4.10) is a generalisation of the result obtained by Aleksander¹ for $D = 2$. An interesting interpretation of (4.10) is that the functional capacity is equal to the number of memory locations in the first layer of the network.

¹ Personal communication, Oct, 88.

4.3. Storage capacity

4.3.1. Definition

The storage capacity of a regular pyramidal WANN is investigated. The storage capacity of the network is defined as a probability distribution, giving the probability $P(p)$ of successful storage of a number p of *arbitrary* input-output associations or patterns.

The number of sets containing p patterns loadable in a pyramidal WANN, with topological parameters N and D , is denoted $S_{N,D}(p)$.

A Boolean function of m variables can be represented as a disjunction of 2^m minterms. If the function is to be learned by a neural network, each minterm defines an binary input pattern and its associated output. The learning task usually consists of storing p such input-output associations. For a network with m inputs, the total number of possible sets of p input-output associations is given by $2^p \cdot \binom{2^m}{p}$.

The probability $P(p)$ that some network can successfully store p *arbitrary* input-output associations can be expressed as the quotient of the number of sets of p input-output pairs that the network is able to store, by the total number of sets of p input-output pairs. For a regular pyramidal WANN, the number of inputs to the network is given by

$$m = N^D$$

and therefore the probability $P(p)$ can be expressed as

$$P_{N,D}(p) = \frac{S_{N,D}(p)}{2^p \cdot \binom{2^{N^D}}{p}}, \quad (4.11)$$

with

$$p = 1, 2, \dots, 2^{N^D}.$$

It is also interesting to notice that the probability $P_{N,D}(2^{N^D})$ that the N , D pyramid can store any set of input-output pattern associations is given by

$$P_{N,D}(2^{N^D}) = 2^{-2^{N^D}} \cdot \Gamma(N, D).$$

in which $\Gamma(N,D)$ represents the functional capacity of the pyramid.

4.3.2. Methodology

The following methodology can be used in order to evaluate the quantity $P_{N,D}(p)$. It consists of setting a count variable $c = 0$ and then repeating n times the procedure:

1. Choose, randomly and without repetition, a set of p distinct input-output pairs, each pair consisting of a binary vector of N^D elements and an associated binary output bit.
2. Evaluate whether the chosen set is loadable by the network. This can be done by one of the following methods:
 - (a). training the pyramidal WANN, using one of the algorithms described in Section 2.3.5.2;
 - (b). using some separability detection algorithm, as described in [A-Ala90];
 - (c). listing the minterms of all the functions computable by the network and check whether any of these functions contain the minterms corresponding to the set of p input-output pairs under consideration.
3. If the chosen set can be loaded in the pyramid, the count variable c is incremented by 1.

At the end of this procedure, the probability $P_{N,D}(p)$ can be evaluated by the quotient

$$P_{N,D}(p) \approx \frac{c}{n}, \quad (4.12)$$

with the evaluation getting better and better as n increases. If the number of iterations in the above process is increased so that all possible sets of p input-output associations are examined, that is,

$$n = 2^p \cdot \binom{2^{N^D}}{p},$$

then it holds that

$$c = S_{N,D}(p)$$

and (4.12) provides *exact* values for the probabilities $P_{N,D}(p)$. This is the method used in the next 2 sub-sections. It should be noted, however, that the above exhaustive search is computationally very expensive and can only be carried out for the smallest networks. In the case where part 2 (c) of the above procedure is used, an evaluation of the cost of the computation is given by:

$$\text{cost} \propto \sum_{j=1}^{2^{N'}} 2^j \cdot \binom{2^{N''}}{j} \cdot \Gamma(N, D). \quad (4.13)$$

For the case $N = 2$, $D = 2$, (4.13) yields

$$\text{cost} \propto 2.238 \cdot 10^{10}.$$

4.3.3. The storage capacity of a regular pyramidal network

A regular pyramidal WANN with parameters $N = 2$ and $D = 2$ is considered. The probability that p arbitrary patterns, or input-output associations, can be stored in the network is computed by exhaustive search. The methodology described in the previous sub-section is employed. For each value of p and for all possible sets of p input-output pairs, it is tested whether all the patterns in the set can be stored in the network. The number of possible sets of p input-output pairs is given by $2^p \cdot \binom{16}{p}$.

The results of the exhaustive computation, performed for the case $N = 2$ and $D = 2$, are given numerically in Table 4.2. The computation took about 60 hours to run on a Sun 3-60 workstation. Figure 4.2 shows a graphical representation of the same results.

It can be noticed that the pyramid can store any set of 4 input-output pairs with probability 1. However, not all sets of 5 pairs can be stored in the network. The following example is considered, in which a set of 5 input-output pairs $T_j = \{i_1, i_2, i_3, i_4; f\}$, with $j = 1, \dots, 5$, are required to be stored in the network. The following input-output associations are chosen:

$$\begin{cases} T_1 = \{0,0,0,0;1\} \\ T_2 = \{0,0,0,1;0\} \\ T_3 = \{0,0,1,0;0\} \\ T_4 = \{1,0,0,1;1\} \\ T_5 = \{1,0,1,0;0\}. \end{cases}$$

Figure 4.1(a) shows a graphical representation of the patterns to be stored in the network represented in Figure 4.1(b). A suffix is used to denote a particular node in the network. For instance, for pattern T_4 , memory locations 10_2 and 01_3 are addressed. The contents of the memory location addressed in node 1 is required to be 1, which is represented graphically by a plain line between the addressed minterms in the bottom nodes. A dashed line is used for the patterns that are required to output a 0.

p	$S_{2,2}(p)$	$2^p \cdot \binom{16}{p}$	$P_{2,2}(p)$
1	32	32	1
2	480	480	1
3	4,480	4,480	1
4	29,120	29,120	1
5	138,624	139,776	$\frac{361}{364} = 0.991758$
6	486,400	512,512	$\frac{950}{1,001} = 0.949051$
7	1,218,560	1,464,320	$\frac{119}{143} = 0.832168$
8	2,071,872	3,294,720	$\frac{10,791}{17,160} = 0.628846$
9	2,482,880	5,857,280	$\frac{38,795}{91,520} = 0.423896$
10	2,175,760	8,200,192	$\frac{135,985}{512,512} = 0.265330$
11	1,411,776	8,945,664	$\frac{22,059}{139,776} = 0.157817$
12	674,528	7,454,720	$\frac{21,079}{232,960} = 0.090483$
13	231,712	4,587,520	$\frac{7,241}{143,360} = 0.050509$
14	54,336	1,966,080	$\frac{283}{10,240} = 0.027637$
15	7,808	524,288	$\frac{61}{4,096} = 0.014893$
16	520	65,536	$\frac{65}{8,192} = 0.007935$

Table 4.2. Storage capacity results for a regular pyramidal WANN with parameters $N = 2$ and $D = 2$

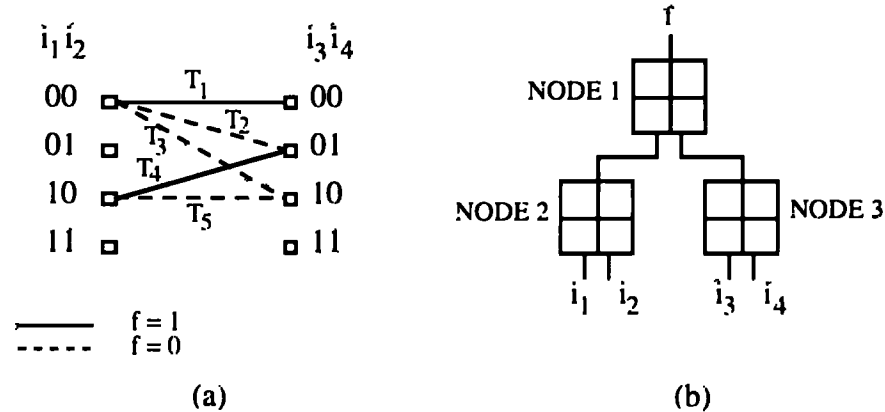


Fig. 4.1: (a) Graphical representation of the 5 input-output associations to be stored in the network; (b) the pyramidal WANN.

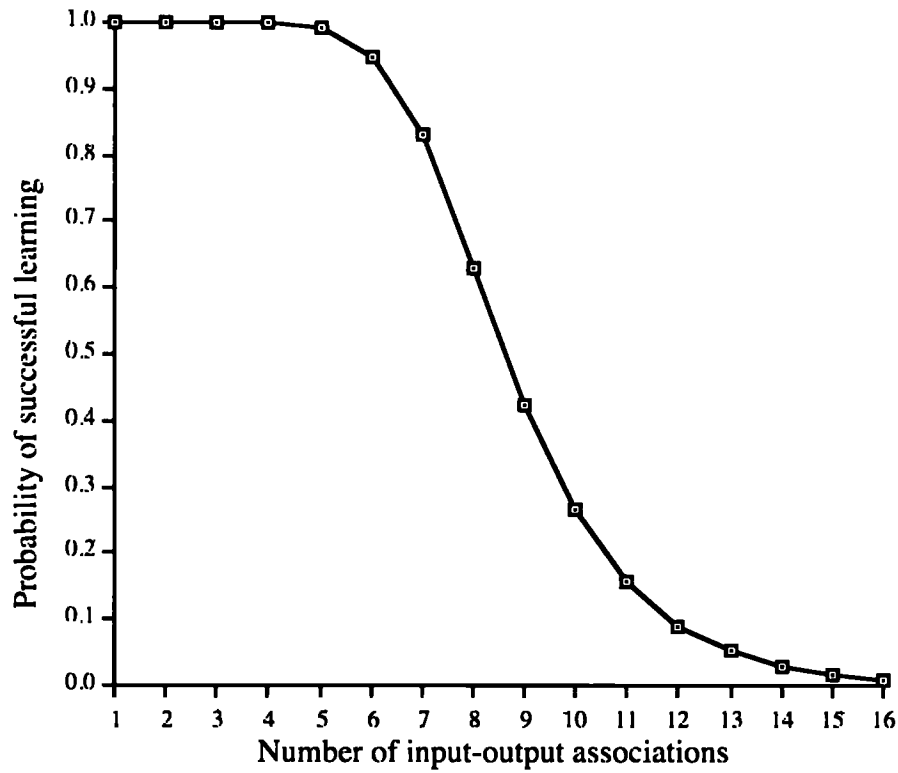


Fig. 4.2: Probability of successful storage as a function of the number input-output associations.

The contents of a memory location xy is denoted $[xy]$. By examining Figure 4.1.(a), it can be seen that

$$T_1, T_2 \Rightarrow [00_3] = [\overline{01_3}]$$

and

$$T_1, T_3 \Rightarrow [00_3] = [\overline{10_3}]$$

and therefore

$$[01_3] = [10_3]. \quad (4.15)$$

Similarly, it can be seen that

$$T_4, T_5 \Rightarrow [01_3] = [\overline{10_3}],$$

which is in contradiction with (4.15) and therefore the input-output pairs (4.14) cannot be

stored in the network.

The above results were first published in [Nto92c]. In [PenS93], subsequently to the work presented in this thesis, a theoretical approximation of $P_{N,D}(p)$ was derived, using the tools of computational learning theory, for the cases corresponding to $N \leq 4$ and $D = 2$. This approximation, however, is over-optimistic compared with corresponding experimental results. For example, for $N = 4$, $D = 2$, it predicts that the pyramid will be able to store any set of 19 input-output associations with 100% probability, whereas experimental evidence shows this number to be of about 8 patterns. Anyhow, the main conclusion remains that the network can only store a very small proportion of the 2^{4^2} (65,536) possible input-output associations.

4.4. Dynamics of learning in pyramidal WANNs

4.4.1. Introduction

The previous sections of this chapter were concerned with functionality and storage capacities of feed-forward pyramidal WANNs. This section deals with the dynamics of learning in the network. In Section 2.3.5.2, several learning algorithms for pyramids were reviewed. The present Section is mainly concerned with the ASA algorithm [Ale89a] and with the parity checking problem.

4.4.2. Previous work

In [Sha89a] (also in [Sha89b]), the dynamics of learning were studied, using an extremely simplified version of the ASA algorithm, which was applied to a pyramidal WANN of 2-input RAM nodes ($N = 2$). The training algorithm consisted of choosing a memory location at random, flipping it and examining how the energy of the system was affected. If the energy decreased, the same operations were repeated by choosing some other memory location at random. If the energy increased the memory location was flipped back to its previous contents. It was shown that the energy surface of the problem is mostly flat with golf course holes which correspond to the solutions. Therefore the task of learning parity can be compared to a random walk. Shapiro calculated an estimation of the number of training steps to solution and showed that it increased exponentially with the number of inputs to the pyramidal WANN.

In [ZhaZZ93], it is also a very simplified version of ASA [Ale89a] which is studied for the case of a 2-layer feed-forward PLN network ($D = 2$). In Aleksander's algorithm, when the output of the network consistently mismatches the desired output, a reset is operated (step 4.2.2 of the algorithm described in Section 2.3.5.2). This reset *only* affects the *addressed* minterms. In [ZhaZZ93], when the output consistently mismatches the desired

output, a *total* reset of the network is operated and the training is started again from the beginning. The method of analysis of the algorithm used in [ZhaZZ93] consists of transforming the simplified PLN training algorithm into a corresponding Markov chain. Then, results of Markov chain theory are applied. The modified training algorithm lends itself to an easier analysis than ASA. In particular, the average number of steps to convergence towards a solution, if this solution exists, is easier to calculate. Indeed, in ASA, it is difficult to determine how many previously learned training patterns are affected by a reset operation.

4.4.3. The parity checking problem

In [Sha89a], the hardness of a problem is related to a low proportion of *solution states* among the possible states of the network's memory. In that respect, the parity checking problem (PCP) belongs to the class of problems which can be classified as 'hardest'. In other words, only a very limited number of memory states of the network solve PCP. In the case of a pyramidal WANN with parameters $N = 2$, $D = 2$, there are only 4 such internal states [Ale89a]. As a thesis contribution, this result is generalised to the following theorem and corollary:

Theorem 4.3: *The number of ways, $F_{PCP}(N, D)$, of computing the parity checking function in a weightless regular pyramidal network with topological parameters N , D is given by*

$$F_{PCP}(N, D) = 2^{\frac{N(N^{D-1} - 1)}{D-1}}, \quad (4.16)$$

Proof: A proof of (4.16) is given in Appendix B.

Corollary 4.1: *The number of ways, $F_{PCP}(N, D)$, of computing the parity checking function in a weightless regular pyramidal network with topological parameters N , D is related to the number $M_{N,D}$ of nodes in the network by the relation*

$$F_{PCP}(N, D) = 2^{M_{N,D} - 1} \quad (4.17)$$

Proof: The proof of (4.17) follows immediately from considering (2.9) and (4.16).

4.4.4. Evolution of the internal state of the network during training

4.4.4.1. Transition probability distributions

The internal state of a pyramidal WANN during training can be seen as the state of a probabilistic automaton [Ale89a]. The state transition graph of this automaton is here studied. The aim is to monitor the dynamic evolution of the number U of u values in the system during the training process, assuming the ASA is used. As in previous sections, the analysis is limited to the case $N = 2$, $D = 2$. Indeed, this case is small enough to enable the derivation of theoretical results.

The learning process can be formalised as that of a particle moving along a one-dimensional line. The particle can be in several positions along the line, corresponding to states from $U = 12$, at the beginning of the training, to $U = 0$, at the end of the training. For any position $U(t) = x$ of the particle, at time step t , there is an associated probability distribution which describes the probabilities of the particle being in the positions

$$U(t+1) = x + \Delta x, \text{ with } \Delta x = -3, -2, \dots, 3,$$

at the time step $t+1$. Therefore, this process can be seen as a *random walk* (e.g., [CoxM65] or [GriS82]), for which positions $U = 12$ and $U = 0$ correspond to *reflecting* and *absorbing* barriers, respectively. However, this random walk is much more complicated than the ones usually studied in the literature on stochastic processes [CoxM65] since it involves a variable probability distribution for each position U of the particle. It is also worth noting that these distributions are averages over several configurations. For example, one configuration could have derived from the state sequence $U(0) = 12$, $U(1) = 9$, $U(2) = 6$, $U(5) = 3$, $U(4) = 8$ and another from the sequence $U(0) = 12$, $U(1) = 9$, $U(2) = 8$. Both configurations have 8 locations set to u . However the corresponding probability distributions of the next number of u locations can be different.

The distributions $p_x(\xi)$ are defined by the expression

$$p_x(\xi) = \Pr(U(t+1) = x + \xi \mid U(t) = x), \quad (4.18)$$

with

$$x = 0, 1, \dots, 12$$

and

$$\xi = -3, -2, \dots, 3,$$

It is also worth noting the expression:

$$\sum_{\xi=-3}^3 p_x(\xi) = 1.$$

A transition matrix can then be written, as shown in Table 4.3. It is worth noting the particular value $p_0(0) = 1$, corresponding to the fact that the state $U = 0$ is analogous to an absorbing barrier. Also, the value $p_{12}(0) = 0$ shows that the state $U = 12$ is a reflective barrier and $p_{12}(-3) = 1$ corresponds to the fact that the first training step always occurs without reset. In the next sub-section, an approximation of the probability distributions (4.18) is calculated.

1	$p_1(-1)$	$p_2(-2)$	$p_3(-3)$	0															0
0	$p_1(0)$	$p_2(-1)$	$p_3(-2)$	$p_4(-3)$	0														
0	$p_1(1)$	$p_2(0)$				$p_5(-3)$	0												
0	$p_1(2)$		$\vdots$				$p_6(-3)$	0											
	$p_1(3)$			ξ				$p_7(-3)$	0										$\vdots$
	0	$p_2(3)$			$\vdots$				$p_8(-3)$	0									
$\vdots$		0	$p_3(3)$			$\vdots$				$p_9(-3)$	0								
	$\vdots$		0	$p_4(3)$			$\vdots$				$p_{10}(-3)$	0							
		$\vdots$		0	$p_5(3)$			$\vdots$				$p_{10}(-2)$	$p_{11}(-3)$	0					
			$\vdots$		0	$p_6(3)$					$\vdots$		$p_{11}(-2)$	1					
0				$\vdots$		0	$p_7(3)$					$\vdots$	$p_{11}(-1)$	0					
0					ξ	0	0	$p_8(3)$	$p_9(2)$	$p_{10}(1)$	$p_{11}(0)$	0							
0	0	0	0	0	0	0	0	0	$p_9(3)$	$p_{10}(2)$	$p_{11}(1)$	0							

Table 4.3: Transition matrix containing the next-state transition probabilities during training with ASA in a regular pyramidal WANN with parameters $N = 2$ and $D = 2$.

4.4.4.2. Calculation of the transition probability distributions

A theoretical calculation is performed for the transition probability distributions (4.18) during training, corresponding to the different U states. The derivation is performed in Appendix C. This yields the following values for the transition probability matrix of Table 4.3:

$$\begin{bmatrix}
 1 & .25 & .029 & .001 & 0 & & & & & & & & 0 \\
 0 & .719 & .439 & .085 & .005 & 0 & & & & & & & \\
 0 & .0 & .487 & .569 & .166 & .015 & 0 & & & & & & \\
 0 & .0 & .0 & .299 & .645 & .309 & .040 & 0 & & & & & \\
 .031 & .0 & .0 & .151 & .575 & .460 & .088 & 0 & & & & & \\
 0 & .044 & .002 & .0 & .076 & .449 & .588 & .170 & 0 & & & & \\
 \vdots & 0 & .043 & .003 & .0 & .035 & .301 & .659 & .329 & 0 & & & \\
 \vdots & & 0 & .030 & .005 & .0 & .013 & .161 & .585 & .524 & 0 & & \\
 \vdots & & & 0 & .020 & .005 & .0 & .004 & .083 & .447 & .751 & 0 & \\
 \vdots & & & & 0 & .012 & .005 & .0 & .001 & .028 & .249 & 1 & \\
 0 & & & & \vdots & 0 & .005 & .003 & .0 & .0 & .0 & 0 & \\
 0 & & & & & \vdots & 0 & 0 & .002 & .002 & .0 & .0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .001 & .001 & 0 & 0
 \end{bmatrix} \quad (4.19)$$

The transition probability distributions (4.18) are represented graphically in Figure 4.3.

4.4.4.3. Convergence of the learning process

The transition matrix (4.19) can be interpreted as the transpose of the transition matrix $\mathbf{P}$ of a Markov chain. In the canonical form, it holds that

$$\mathbf{P} = \begin{matrix} & \begin{matrix} A & T \end{matrix} \\ \begin{matrix} A \\ T \end{matrix} & \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{R} & \mathbf{Q} \end{bmatrix} \end{matrix},$$

in which A and T represent the absorbing and transient state sets, $\mathbf{Q}$ represents transition probabilities within T and $\mathbf{R}$ represents transition probabilities from T to A . Therefore, using the values in matrix (4.19), it holds that:

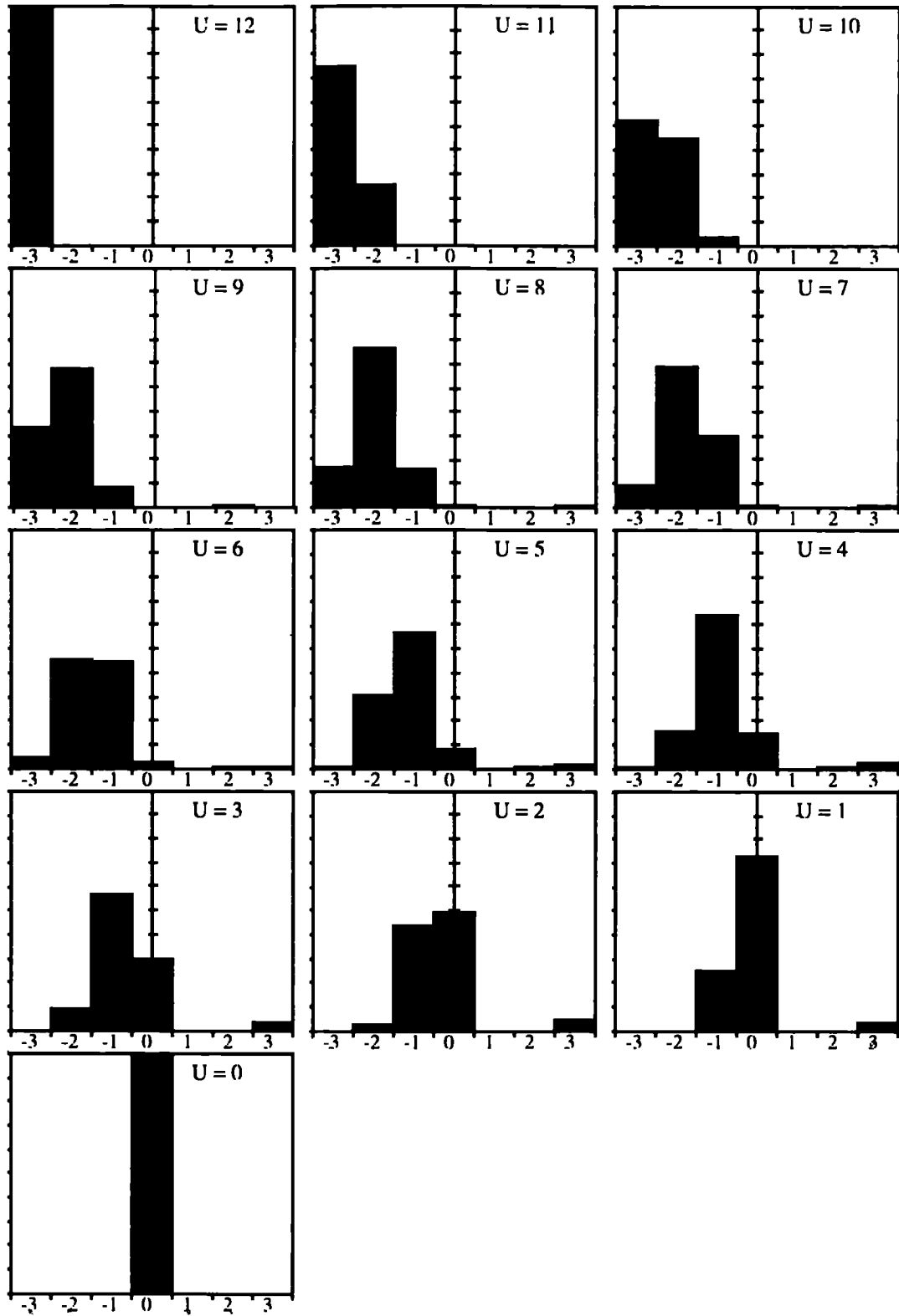


Fig. 4.3: Next-state transition probability distributions during training to PCP, with ASA, in a regular pyramidal WANN with parameters $N = 2$ and $D = 2$.

$$Q = \begin{bmatrix} .719 & .0 & .0 & .031 & 0 & 0 & & & & & & 0 \\ .439 & .487 & .0 & .0 & .044 & 0 & & & & & & 0 \\ .085 & .569 & .299 & .0 & .002 & .043 & 0 & & & & & \\ .005 & .166 & .645 & .151 & .0 & .003 & .030 & 0 & & & & \\ 0 & .015 & .309 & .575 & .076 & .0 & .005 & .020 & 0 & & & \\ 0 & 0 & .040 & .460 & .449 & .035 & .0 & .005 & .012 & 0 & & \\ & & 0 & .088 & .588 & .301 & .013 & .0 & .005 & .005 & 0 & 0 \\ \vdots & & & 0 & .170 & .659 & .161 & .004 & .0 & .003 & .002 & 0 \\ & & & & 0 & .329 & .585 & .083 & .001 & .0 & .002 & .001 \\ & & & & & 0 & .524 & .447 & .028 & .0 & .0 & .001 \\ 0 & & & & & & 0 & .751 & .249 & .0 & .0 & 0 \\ 0 & 0 & & & & & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

and

$$R = [.25 \ .029 \ .001 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]^T.$$

A calculation similar to that reviewed in Section 2.5.3.2.1 can be carried out and matrices N and V can be calculated using (2.13) and (2.14):

$$N = (I - Q)^{-1}$$

$$N = \begin{bmatrix} 3.9725 & .2337 & .1617 & .1656 & .0198 & .0097 & .0054 & .0005 & .0001 & .0 & .0 & .0 \\ 3.7314 & 2.3328 & .2736 & .2355 & .1267 & .0175 & .0084 & .0027 & .0002 & .0001 & .0 & .0 \\ 3.7559 & 2.0586 & 1.7659 & .2900 & .1515 & .0854 & .0108 & .0036 & .0010 & .0001 & .0 & .0 \\ 3.7523 & 2.1029 & 1.4554 & 1.4908 & .1779 & .0877 & .0481 & .0043 & .0012 & .0003 & .0 & .0 \\ 3.7533 & 2.0918 & 1.5412 & 1.0605 & 1.2694 & .1040 & .0438 & .0265 & .0014 & .0003 & .0001 & .0 \\ 3.7530 & 2.0957 & 1.5093 & 1.2354 & .6961 & 1.1425 & .0528 & .0216 & .0134 & .0004 & .0001 & .0 \\ 3.7531 & 2.0940 & 1.5237 & 1.1533 & .9940 & .4250 & 1.0663 & .0259 & .0101 & .0058 & .0001 & .0 \\ 3.7530 & 2.0948 & 1.5172 & 1.1919 & .8434 & .8464 & .2181 & 1.0304 & .0115 & .0044 & .0022 & .0 \\ 3.7531 & 2.0948 & 1.5184 & 1.1836 & .8831 & .6972 & .6605 & 1.093 & 1.0134 & .0039 & .0022 & .0006 \\ 3.7529 & 2.0942 & 1.5205 & 1.1714 & .9234 & .6213 & .6752 & .4777 & .0394 & 1.0051 & .0011 & .0008 \\ 3.7531 & 2.0948 & 1.5175 & 1.1899 & .8533 & .8092 & .3285 & .8006 & .2614 & .0043 & 1.0022 & .0002 \\ 3.7531 & 2.0947 & 1.5184 & 1.1836 & .8831 & .6972 & .6605 & 1.093 & 1.0134 & .0039 & .0022 & 1.0006 \end{bmatrix}$$

and

$$V = N \cdot R = [1.0 \ 1.0 \ 1.0 \ 1.0 \ 1.0 \ 1.0 \ 1.0 \ 1.0 \ 1.0 \ 1.0 \ 1.0 \ 1.0]^T \quad (4.20)$$

Equation (4.20) shows therefore that the probability that any state, and in particular state $U = 12$, converges towards state $U = 0$ is equal to 1. This demonstrates the convergence of the training algorithm in the particular case considered.

The mean number of pattern presentations for any state $U = i$ to converge towards $U = 0$ is given by

$$T_i = \langle \tau(i) \rangle = \sum_{j=1}^{12} N_{ij},$$

which in matricial notation yields

$$\mathbf{T} = [4.57 \ 6.73 \ 8.12 \ 9.12 \ 9.89 \ 10.52 \ 11.05 \ 11.51 \ 11.92 \ 12.28 \ 12.62 \ 12.92]^T,$$

4.5. Conclusions

The chapter was concerned with feed-forward pyramidal WANNs. Results about the functional capacity, the storage capacity and the learning dynamics of such networks have been obtained.

An exact and non-recursive formula was derived for the functional capacity of a D -layer pyramidal network with 2-input nodes ($N = 2$), and it was shown that the functional capacity of the network grows as 6^{2D} . An approximation to the functional capacity of a pyramid with general parameters N , D was derived.

A methodology was proposed in order to investigate the storage capacity of the network, defined as the number of arbitrary input-output associations that it can store. A exact probability distribution was obtained, by exhaustive search, for the case $N = 2$ and $D = 2$.

The ASA training algorithm was considered and the learning dynamics of the pyramidal network, trained on the parity checking problem, were investigated. The number of solutions to the parity problem was calculated, for general parameters N and D . A calculation of the transition probabilities of the pyramid's internal state, during training of the PCP, was carried out for the case $N = 2$, $D = 2$. The memory contents of the network, trained under the ASA, to the PCP, was proved to converge towards a fully trained configuration.

CHAPTER V, UNSUPERVISED LEARNING IN WEIGHTLESS NEURAL NETWORKS

5.1. Introduction

Physiological studies have shown that sensory information is encoded by the brain, at least in the primary sensory areas, in various geometrically organised maps [Koh88a]. Sensory experiences such as topographic co-ordinates of the body or pitch of audible tones are mapped into a two-dimensional co-ordinate system defined on some areas of the cortex [Cha83]. These areas are composed of two-dimensional layers of neurons in which there is dense lateral interconnection with relatively few interlayer connections [Koh89a]. The degree of lateral interaction between cells is usually described as having the form of a Mexican hat. Three types of lateral interaction can be distinguished: a short range lateral excitation surrounded by a penumbral region of inhibitory action which is surrounded itself by a weaker excitatory action [Koh89a].

In [Koh82a] (also in [Koh82b] or [Koh82c]), it is shown that a self-organising array of linear neurons is capable of forming a topologically ordered map of data to which it has been exposed, automatically forming a reduced representation of input information. This paradigm was successfully applied to speech recognition (e.g., [Koh88b] or [Koh89b]).

In this chapter, a network with similar properties but using weightless nodes is defined. The first node model to be considered is the pyramidal weightless neural node, reviewed in Section 2.3.5 and further studied in Chapter 4. A brief discussion is made about the reasons why this model is inadequate for unsupervised training under a Kohonen-type training scheme. Then, as the main contribution of this chapter, a new unsupervised training algorithm is introduced, involving the use of a spreading procedure. The C-discriminator node model, or continuously-valued discriminator (CDN) is defined. It is shown, on a first example, that the behaviour of the network can be predicted considering the overlap areas between input patterns. A second example follows which demonstrates the network behaviour under a uniform input distribution. Finally, the CDN model is compared to other weightless systems involving unsupervised learning.

5.2. Pyramidal nodes

The pyramidal weightless node was first considered as a weightless node model to be used in a kohonen-type network. When an input pattern is presented to an array of pyramidal

nodes, each pyramid in some region of the array can be trained to output a value 1. When the same (or similar) input pattern is subsequently presented to the network, the considered group of pyramids is expected to output a value 1 again. Similarly, other groups of pyramids can be trained to output 0 when presented with particular input patterns.

When a linear weighted-sum neuron (LU) is trained with some input pattern, the LU's weight vector is modified such that it gets closer, in weight space, to the input pattern. Furthermore, the weight vector is 'pushed away' from patterns dissimilar to the input pattern. Figure 5.1 shows a graphical representation of this behaviour, in the case of two-dimensional input and weight spaces. Vectors i_1 and i_2 represent two training patterns. The vectors w and w' represent the neuron weights before and after training to pattern i_2 , respectively.

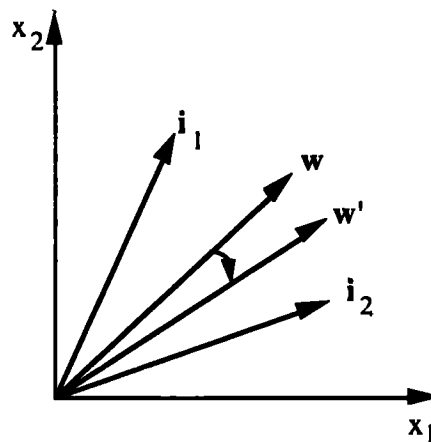


Fig. 5.1: Graphical representation of the change in weights of a 2-input linear neuron during training to input pattern i_2 . The vectors w and w' represent the neuron weights before and after training, respectively.

A similar behaviour would be desirable with a pyramidal node. However, this is not what happens; when a pyramidal node is trained to respond 1 to some input pattern, the response of the node to a very dissimilar pattern will not necessarily be 0. This behaviour is a consequence of the poor generalisation ability of pyramids, which was reviewed in Section 2.3.5.4. Another major problem encountered when trying to use pyramids as the nodes of a Kohonen-type network, is their low storage capacity. The storage capacity of a pyramidal node was defined, in Section 4.3.1, as the number of arbitrary input-output associations that the node can store. In practice, a low storage capacity means that, as new training patterns are being loaded in the pyramidal node, older ones get disrupted.

5.3. Discriminator-based nodes

5.3.1. Introduction

In this Section, a neural network showing similar self-organising properties to a Kohonen network is demonstrated. A discriminator-based weightless neuron model is used, whereas the nodes of a standard Kohonen network are linear weighted-sum neurons. There are two main differences between the discriminator model used in this chapter, hereafter referred to as C-discriminator node (CDN), or continuously-valued discriminator, and the model presented in Section 2.3.3, referred to as discriminator node (DN).

Firstly, the storage locations in the discriminator's RAMs hold binary values (1's and 0's) whereas in the CDN they are allowed to store a continuous range of values between 0 and 1. This assumption is made in order to remain as general as possible. Obviously, approximations are possible, for example, as a range of discrete values in the interval [0,1]. In the following, the RAMs of the CDN are referred to as C-RAMs (Continuously-valued RAMs).

Secondly, each DN of a network is usually assigned to a certain class of input patterns, as in the single layer discriminator network (SLDN) reviewed in Section 2.5.2.2, whereas in the present system each input pattern is seen by all CDNs during the training of the system. In fact, the class to which the presented patterns belong is not known *a priori* and, in that sense, the training is seen as *unsupervised* as opposed to a *supervised* training in the SLDN.

5.3.2. The discriminator-based network

5.3.2.1. The network

The overall structure of the network considered here is similar to that of a Kohonen network. The network is made of one layer of processing units or neurons which are modelled as C-discriminators. The units are labelled

$$C_1, C_2, \dots, C_j, \dots, C_Q$$

and are arranged as a one or two-dimensional array. An example of the one-dimensional arrangement is shown in Figure 5.2.

An important characteristic of this structure is that each C-discriminator sees the whole of any input pattern presented to the network. The input patterns consist of arrays of m black or white pixels which are respectively translated into 1's and 0's. Each C-

discriminator produces an output response r and the set of responses yields an output map. During training of the network, the neurons are interacting with each other through lateral excitation or inhibition.

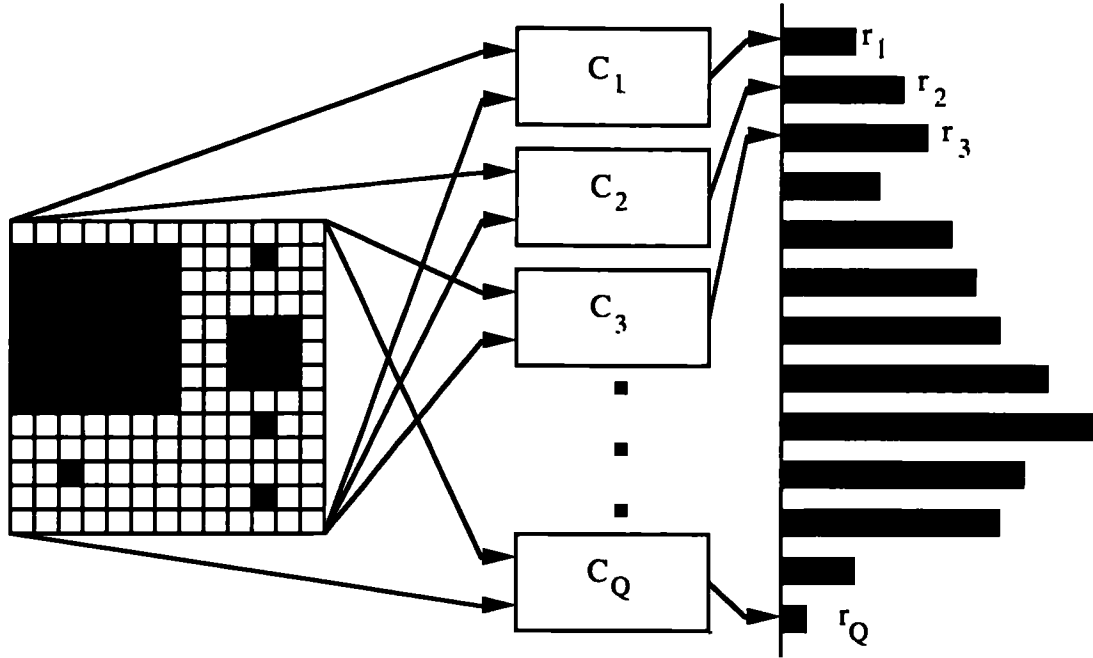


Fig. 5.2: The C-discriminator neural network.

5.3.2.2. The C-discriminator node

The structure of a C-discriminator is shown in Figure 5.3. It is an array of C-RAMs whose memory locations can hold a continuous range of values between 0 and 1. The number K of such C-RAMs in a C-discriminator is

$$K = \frac{m}{n},$$

n being the n -tuple size and m the number of inputs to the C-discriminator. The value m is a multiple of n . The output f_k of a RAM k takes real values between 0 and 1. The binary vector

$$\mathbf{i}_k = (i_1, \dots, i_n)_k$$

is used to denote both the input address to C-RAM k and the storage location addressed by it. The value stored in a memory location with address $\mathbf{a} = (a_1, \dots, a_n)$ is denoted $[\mathbf{a}]$ or $[a_1, \dots, a_n]$. Hence, it holds that:

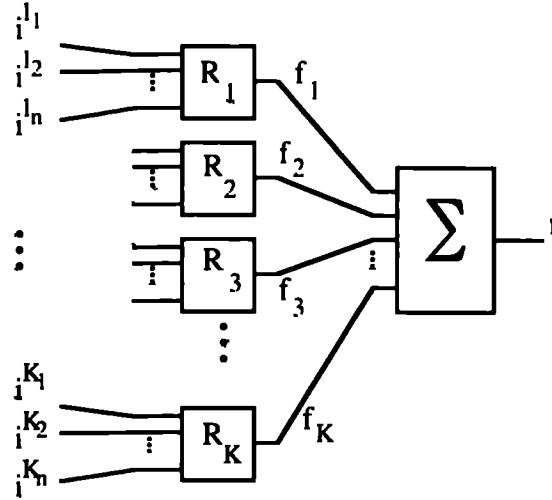


Fig. 5.3: Structure of a C-discriminator.

$$f_k = [i_1, \dots, i_n]_k.$$

The output r_j of a discriminator C_j can be written as

$$r_j = \frac{1}{K} \sum_{k=0}^K f_k^j = \frac{1}{K} \sum_{k=0}^K [i_1, \dots, i_n]_k^j \quad (5.1)$$

with $0 < r_j < 1$. Hereafter r_j will be quoted as a percentage.

5.3.3. An unsupervised training algorithm

The training procedure follows the rules that form the basis of various self-organising processes. When a pattern is presented to the network, the best matching neuron is located, that is, the one which responds with the highest value. Then, the matching is increased at this neuron and its topological neighbours, and decreased at more distant neurons. This procedure is now examined in detail.

First, the memory content of the neurons is initialised at random values within the interval $[0,1]$. Then, an input pattern, chosen randomly amongst a set of training patterns, is presented to the network. All the C-discriminators see that pattern simultaneously. The neuron F whose response r_F to the pattern is the highest is said to be *firing*. That means that this neuron is the one which matches best the input pattern and it holds that

$$r_F = \max_{j=1, \dots, Q} (r_j),$$

An excitation and an inhibition regions of interaction about neuron F are defined. The spatial extents of these regions are respectively characterised by an excitation radius ρ_{ex} and an inhibition radius ρ_{inh} . The memory content of the neurons in these regions is updated in such a way that the neurons lying in the excitation region become more susceptible to fire when the same or a similar pattern is presented again to the network, while the neurons in the inhibition region are 'pushed away' in terms of their response to the input pattern. All other neurons in the network are left unchanged. Also, during training, Kohonen's principle of 'shrinking' neighbourhoods [AllBJ89] is applied, that is, the excitation and inhibition radiuses are treated as monotonically decreasing functions of the number of training pattern presentations.

A particular C-RAM R_i is considered, in a C-discriminator situated in the excitation region of neuron F_* . The Hamming distance between the input i_i to R_i and the address a_i of a certain memory location to be updated in R_i is denoted by H_u . Every memory location a_i in R_i will be updated according to the relation

$$[a_i]' = [a_i] + k \cdot |\Delta| \cdot (\tau(\Delta) - [a_i]) \quad (5.2)$$

with

$$\Delta = \frac{n}{2} - H_u, \quad (5.3)$$

τ being the Heaviside function defined as

$$\tau(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \quad (5.4)$$

$|\Delta|$ being the absolute value of Δ , τ denoting an updated value and k being an *excitation coefficient*, with $0 < k < 1$. Similarly for a neuron in the inhibition region.

$$[a_i]' = [a_i] + l \cdot |\Delta| \cdot (\tau(-\Delta) - [a_i]), \quad (5.5)$$

l being an *inhibition coefficient*, with $0 < l < k$. In many simulations the neural density is insufficient to include a region of inhibition. The general nature of the process is similar for a wide range of values k and l . It is mainly the speed of the process which varies.

Following the updating phase, another input pattern is selected and presented to the network, and the training process starts again. The number of training iterations is problem dependent and can be as small as 1,000, as in the example presented in Section 5.4.1. One of the most central phenomena associated with this training procedure is a clustering effect

that makes similar patterns to produce output responses close to one another in the network output map.

5.3.4. Explanation of the Equations (5.2) and (5.5)

Equation (5.2) is the only one considered here, as a similar reasoning can be applied to (5.5). Expression (5.2) describes the update operation taking place in any C-RAM R_i of a C-discriminator in the excitation region. In the sequel, the subscript i is omitted. All memory locations in R are updated. The vector $\mathbf{i}$ denotes the input pattern to R ; it is the part of the overall input pattern which feeds into R . The vector $\mathbf{a}^j$ denotes the address of a memory location j (or the memory location itself). Hence, the expression

$$H_a = d_H(\mathbf{i}, \mathbf{a}^j).$$

The update will be as follows. The contents $[\mathbf{a}^j]$ of a memory location $\mathbf{a}^j$ will be increased if H_a is small and decreased if H_a is large. The increase or decrease is proportional to the value H_a . Furthermore, in order for the contents of a memory location to stay within the range $[0,1]$, the increase or decrease of the contents of a memory location will be made to be proportional to the 'distance' between the target value (1, for $H_a < \frac{n}{2}$, or 0, for $H_a > \frac{n}{2}$) and the current stored value. This can be expressed as

$$[\mathbf{a}^j]' = [\mathbf{a}^j] + k \cdot \left(\frac{n}{2} - H\right) \cdot (1 - [\mathbf{a}^j]), \quad \text{for } 0 \leq H_a \leq \frac{n}{2} \quad (5.6)$$

and

$$[\mathbf{a}^j]' = [\mathbf{a}^j] - k \cdot \left(H - \frac{n}{2}\right) \cdot ([\mathbf{a}^j] - 0), \quad \text{for } \frac{n}{2} < H_a \leq n. \quad (5.7)$$

Equations (5.6) and (5.7) reduce to the single expression (5.2), after substitutions of (5.3) and (5.4).

5.3.5. Choice of a linear spreading function

The algorithm for the update of the C-RAM contents, following the presentation of an input pattern, can be viewed as a *spreading* algorithm (see Section 2.4.3). Indeed, the information in the addressed memory location is spread throughout the whole of the node memory. This process affects memory locations not addressed by the training pattern. The

semi-linear spreading function $\left| \frac{n}{2} - H \right|$ is used, as expressed by the factor $|\Delta|$ in (5.2) and (5.5), and represented in Figure 5.4.

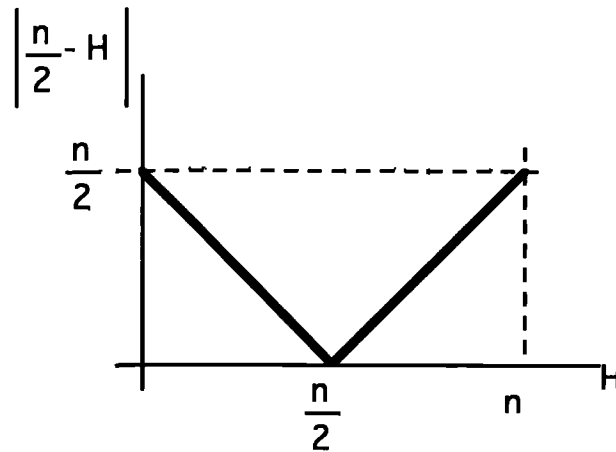


Fig. 5.4: Spreading function.

The reason for this choice of spreading function is best explained by the following example. A 4-input RAM is considered. The input to the RAM is denoted by the 4-tuple (i_0, i_1, i_2, i_3) , e.g. (1,0,0,1). The input element i_0 is considered first. Its value (1 in this example) contributes to the possible addressing of 8 memory locations, the addresses of which have their first bit set to 1. These locations are conceptually tagged with +1. Similarly, the remaining locations are tagged with -1 to indicate that they are unlikely to be addressed, because the first bit of their addresses has the value 0. This labelling is repeated for input elements i_1 , i_2 and i_3 . This is represented on a Karnaugh map in Figure 5.5.

The combined effect of all 4 inputs is accounted for by adding together the contents of the Karnaugh maps. This is represented in Figure 5.6 (a). The addressed location (1001) contains the highest value, +4. The opposite location (0110) contains the smallest value, -4. Figure 5.6 (b) shows a graph of the variation of the location contents with respect to the Hamming distance from the addressed location. By normalising this variation with respect to n , which is the maximum Hamming distance possible between any 2 locations, and by taking the absolute value, the graph of Figure 5.4. is obtained. The role of this spreading function can be seen as a way of 'linearising' the C-RAM function.

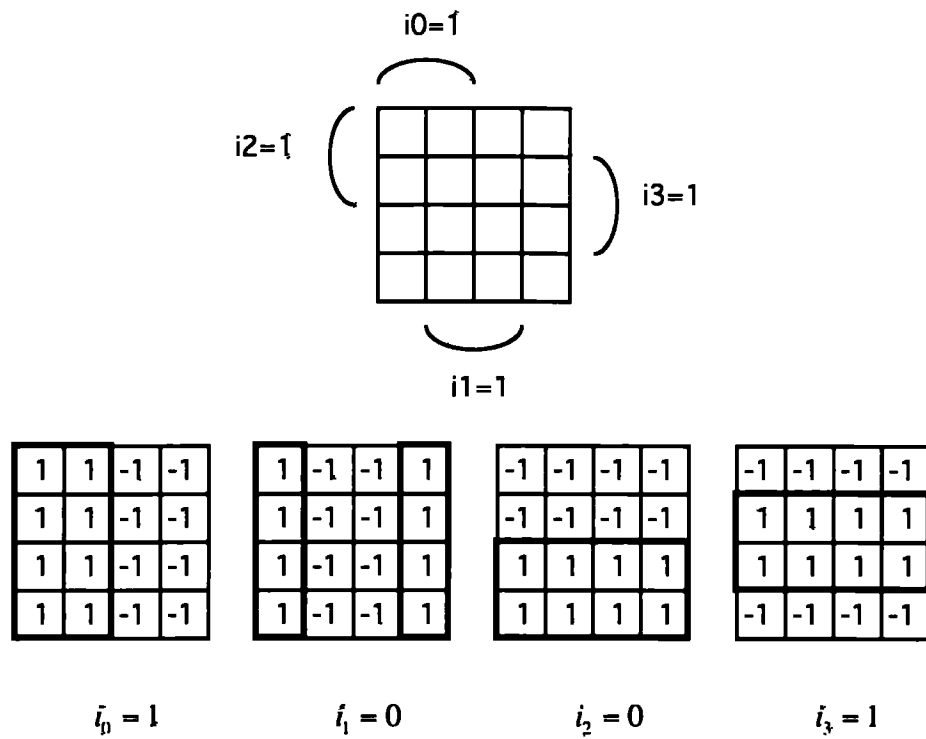


Fig. 5.5: Representation in a Karnaugh map of the effect of each input bit on the addressing of a memory location in a 4-input RAM.

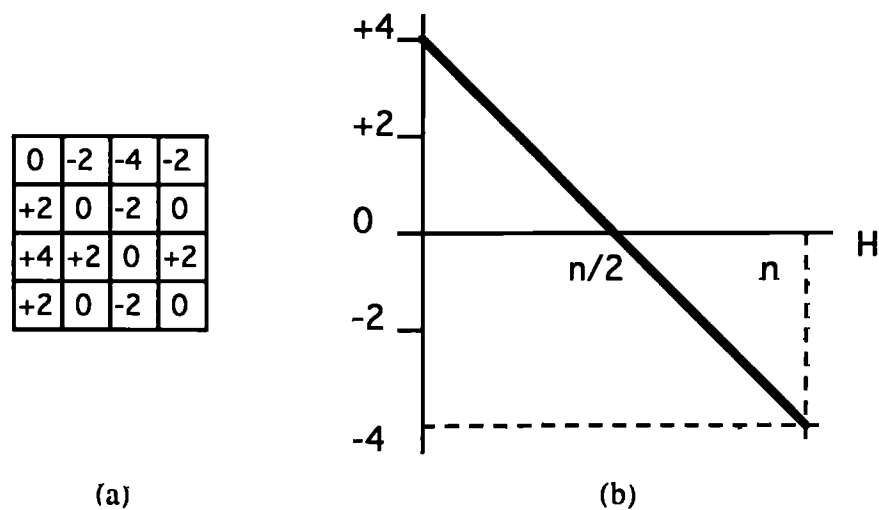


Fig. 5.6: Combined effect of the inputs on the addressing of a memory location in a 4-input RAM.

5.4. Experimental results

5.4.1. Simulation 1

5.4.1.1. Introduction

In this first example, it is shown that the C-discriminator network can form a topologically ordered map of input data, resulting in a *clustering* of responses to similar patterns in specific regions of the output map. It is argued that it is possible to predict the network responses to different input patterns considering the overlap areas between them. The temporal evolution of responses to a certain pattern is shown. The formation of specific areas of activity in the output map is observed. A simulation is also performed using a standard Kohonen network trained with the same input patterns and the resulting output map is compared with the map generated by the C-discriminator network.

5.4.1.2. The simulation

The problem given to the network is to locate the position, in the input pattern, of a black area, here represented as a square. The training patterns form four classes. In each of them, the black square is positioned at one of the corners of the patterns. Patterns belonging to the same class differ by a certain number of pixels chosen randomly and whose colour has been inverted to the opposite one. This acts as an addition of noise to the four basic patterns. In the simulation, up to 4 pixels were flipped in an input pattern. Each input pattern consists of an array of 8 by 8 pixels. Therefore, there are 64 inputs fed into each of the network C-discriminators. Typical training patterns from the 4 classes are shown in Figure 5.7.

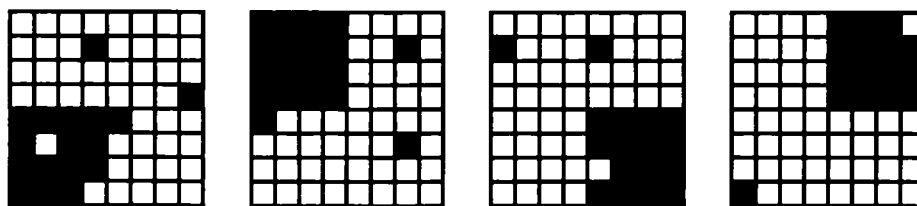


Fig. 5.7: Typical training patterns.

The network is made of 40 C-discriminators arranged in a one-dimensional array. This yields a one-dimensional map of responses. The size of the n -tuple is chosen to be 4. Consequently, there are 16 RAMs in each C-discriminator.

The inhibition interactions between neurons are neglected. The radius of excitation is

treated as a decreasing function of the number of input patterns seen, starting at a value of 10 and decreasing by 1 after each 100 training steps. Although this decreasing radius improves moderately the response clusterings, this is not a critical factor in the behaviour of the network, compared to a fixed radius. It may be argued that decreasing the radius of the excitation region, as learning proceeds, is somewhat analogous to the process of annealing in Boltzmann nets. The excitation coefficient k is chosen equal to 0.1.

The test set consist of 9 patterns, labelled $P_1, P_2, \dots, P_9$. Patterns P_1 to P_4 are the prototypes of patterns seen by the network during training, whereas patterns P_5 to P_9 are prototypes to intermediate classes of patterns never seen by the network before. Patterns P_5 to P_9 are represented in Figure 5.8.

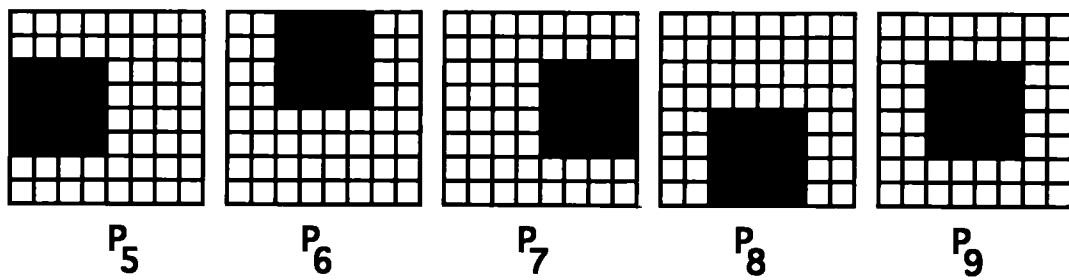


Fig. 5.8: Prototype test patterns P_5 to P_9 .

In Figure 5.9., the network responses after 1000 training steps are shown. Although it is difficult to derive mathematically the responses given by the network, starting from (5.2), it is possible to justify them considering the principle of 'overlap areas' on which the discriminator node is based. The network discriminators are grouped into four sets $S_1, \dots, S_4$, such that discriminators belonging to set S_i give approximately 100 % response when presented with the pattern P_i . Then, the response $r_{S_i}(P_j)$ of a discriminator, belonging to set S_i , to a pattern P_j is given by the overlap area between patterns P_i and P_j :

$$r_{S_i}(P_j) \approx 100 \cdot \omega_{ij}, \quad (5.8)$$

defining ω_{ij} as the fraction of identical pixels in patterns P_i and P_j . Equation (5.8) can be interpreted as being equivalent to (2.6), considering that, in (5.8), the discriminator responses are expressed as percentages and that the exponent n has been eliminated by the linearising effect of the spreading function used during training (see Section 5.3.5). The overlap area between each pairwise combination of the patterns P_1 to P_4 is 50 %, observed to be a reasonable prediction of the responses obtained in the simulation results of Figure 5.9. Considering the case of an intermediate pattern, for example P_5 , it has 75 % overlap area with patterns P_1 and P_2 , and 50 % with patterns P_3 and P_4 . Pattern P_9 produces

approximately 62.5 % response at each C-discriminator, which is the overlap area between P_0 and each of the patterns P_1 to P_4 .

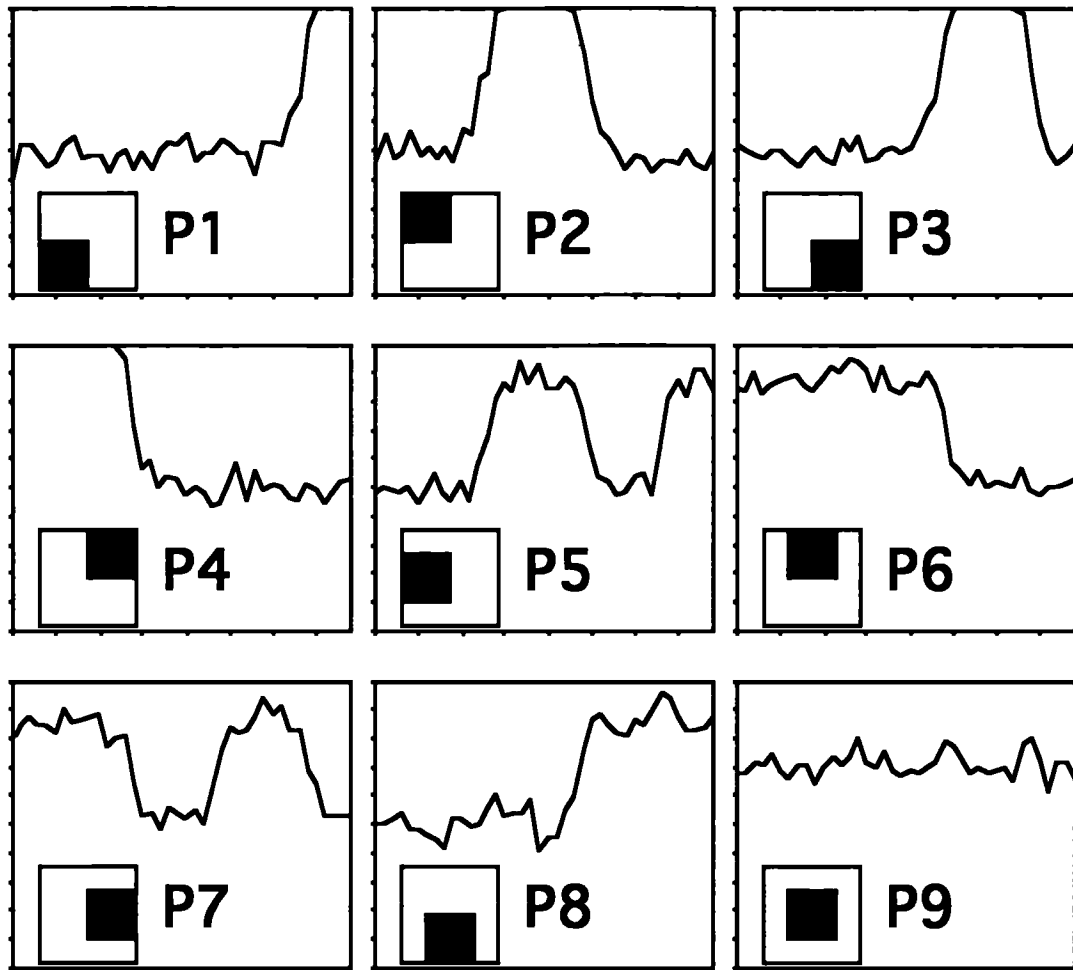


Fig. 5.9: Responses to the test patterns P_1 to P_9 at time $T = 1000$. $Exc.=10$, $Inh.=0$, Noise: 0 to 4 pixels.

In the above simulation, the excitation radius was chosen to start at a value of 10 and was progressively decreased during training. In Figures D.1 to D.3 of Appendix D.1, the network responses after 1000 training steps are shown, for excitation radiuses taking the values 2, 5, 10 and 20, and kept constant during training. These results show that the value of the excitation radius affects only slightly the extent of the areas of activity in the output map, with larger radiuses producing larger areas of activity. These results also show that decreasing of the excitation radius during training is not a critical factor in the behaviour of the network.

The variability between patterns belonging to the same class was simulated by randomly choosing a certain number of pixels, defined as a noise level, from one of the

basic training patterns, P_1 to P_4 , and inverting their colour. In Figures D.4 to D.6 of Appendix D.2, the network responses after 1000 training steps are shown, for noise levels taking the values 0, 4, 10, 20 and 32. These results show that, as the noise level is increased from 0 to 4, the network responses to the test patterns form sharper regions of activity. A further increase of the noise level does not significantly affect the network output map.

5.4.1.3. Temporal evolution of responses

The temporal evolution of neuron responses, to a certain pattern, at different times throughout the training procedure is shown in Figure 5.10.

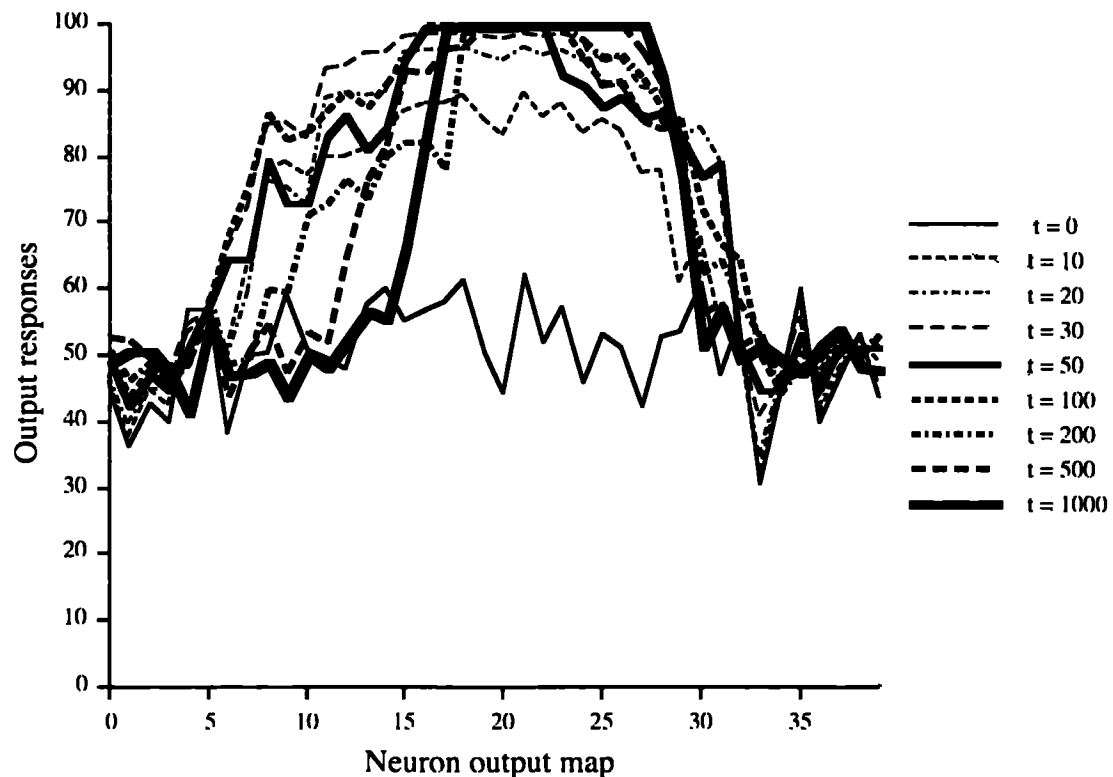


Fig. 5.10: Development of responses to a particular pattern over time. The responses at times $T = 0, 10, 20, 30, 50, 100, 200, 500$ and 1000 are shown.

Initially, the C-discriminator which outputs the strongest response for a particular pattern is randomly distributed in the net. As learning proceeds, a 'bubble' of response begins to emerge in a certain region of the output map. Then, the clustering of activity stabilises to produce a stable output map of responses. Figure 5.11 shows the temporal evolution of responses for prototype input patterns P_1 to P_4 during training.

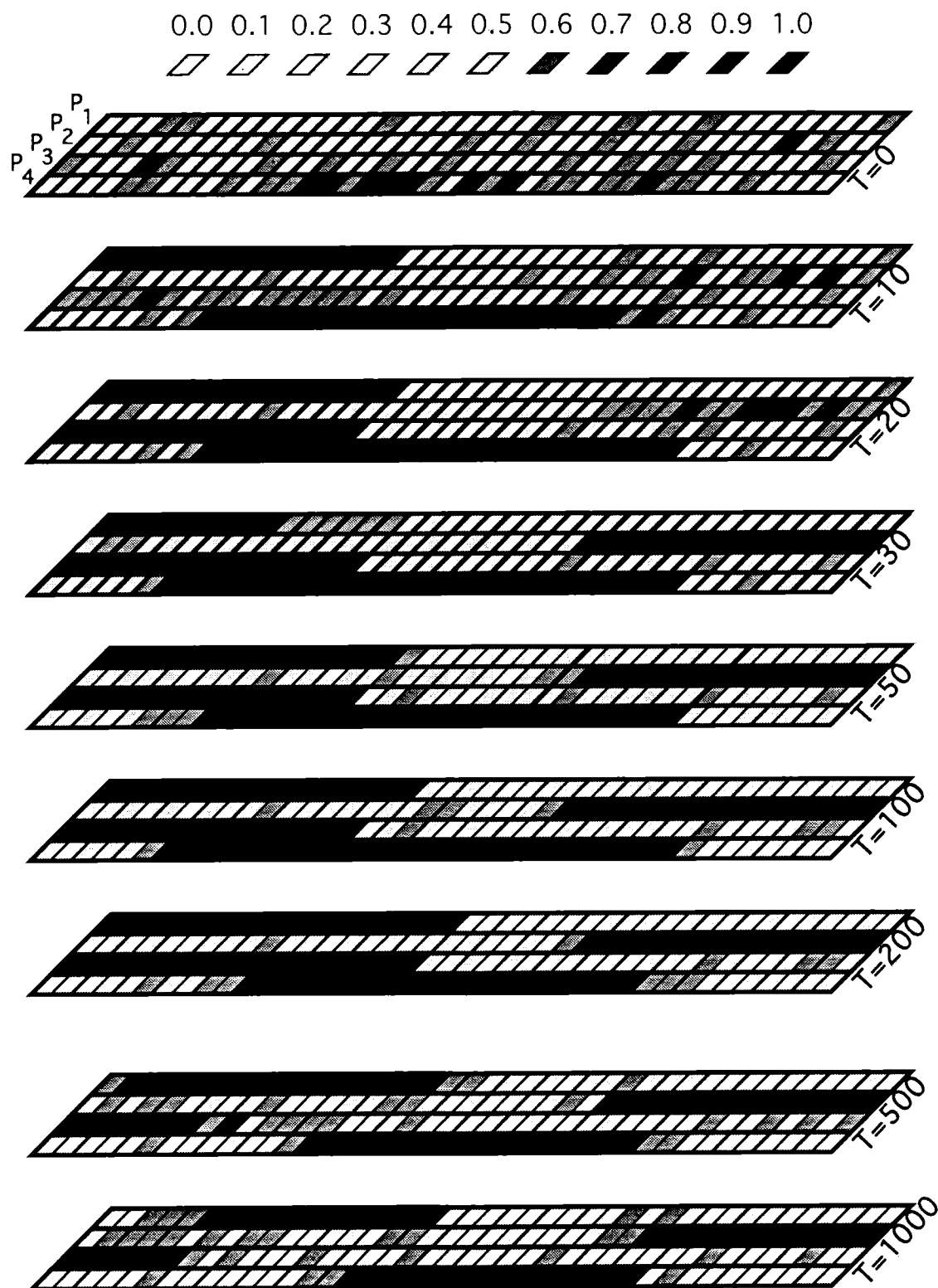


Fig.5.11: Development of responses to prototype patterns P_1 to P_4 over time.

5.4.1.4. Comparison with a standard Kohonen network

The simulation described in Section 5.4.1.2, in which the network nodes were C-discriminators, was also performed with a standard Kohonen network, in which the nodes consisted of linear weighted-sum neurons. The black and white pixels of the training patterns were translated into the values $x_i = 1$ and $x_i = -1$, taken by the inputs to the neurons. The weights w_{ji} of the network were allowed to take real values in the range $[-1, 1]$. Hence, the output response of a neuron could be expressed as

$$y_j = \sum_{i=1}^m w_{ji} \cdot x_i, \quad (5.9)$$

m being the number of inputs per neuron and with y_j taking values in the range $[-m, m]$.

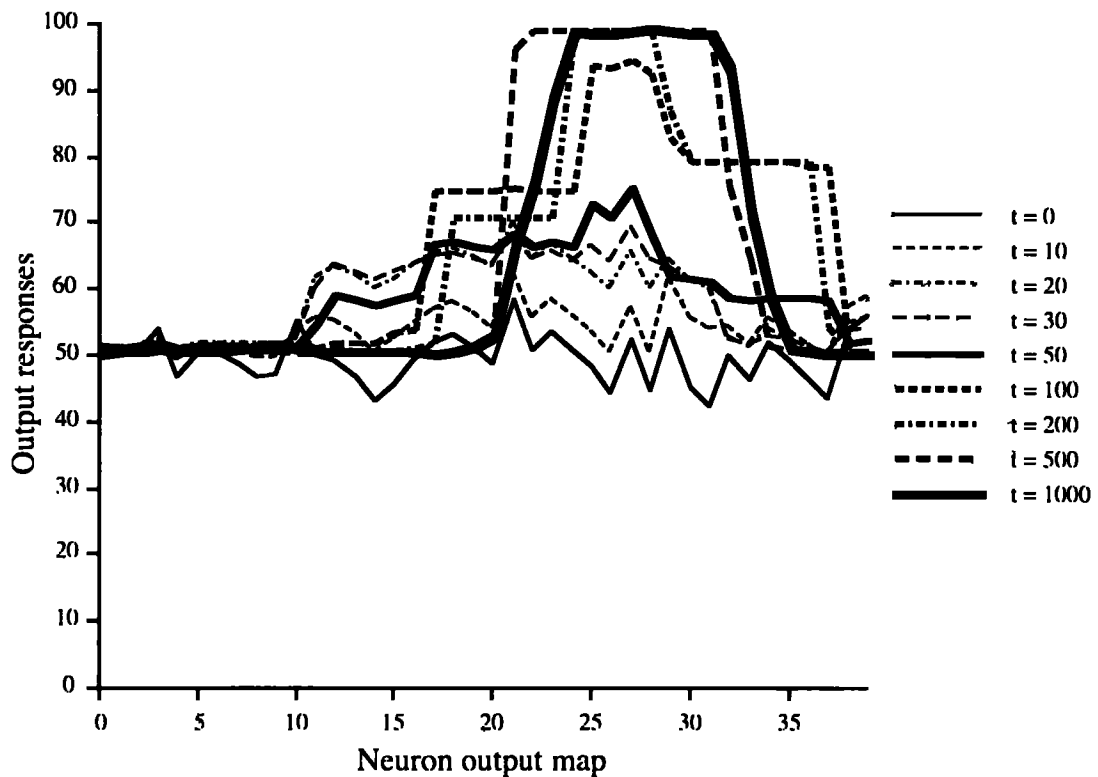


Fig. 5.12: Temporal evolution of responses in a standard Kohonen network with linear weighted-sum neurons.

In order that the output response of a neuron takes a value in the range $[0, 1]$, as in the C-discriminator network, (5.9) is modified to:

$$y_j = \frac{1}{2} + \frac{1}{2m} \cdot \sum_{i=1}^m w_{ji} \cdot x_i.$$

During training, the weights of the neurons situated in the excitation region of the network are updated according to the relation:

$$w'_{ji} = w_{ji} + k \cdot (x_j - w_{ji}).$$

The temporal evolution of the network responses to one of the basic patterns, say P_1 , is shown in Figure 5.12. These results can be compared with those obtained in Figure 5.10 for the C-discriminator network. In both cases, a 'bubble' of response begins to emerge in a certain region of the output map, which stabilises to produce a stable output map of responses. The formation of the bubble of activity would appear to be faster in the C-discriminator network than in the corresponding Kohonen network, although, due to the very limited size of the simulation, no definitive conclusion can be drawn. The main conclusion here is that both networks exhibit a similar behaviour in response to the same training data.

5.4.2. Simulation 2: uniform input pattern distribution

In this section, the self-organising properties of a C-discriminator network are demonstrated in the case of a uniform distribution of the binary variables forming the inputs to its nodes. More specifically, a C-discriminator network of Q^2 nodes is considered, in which the nodes are arranged in a 2-dimensional array. The node outputs form therefore a $Q \times Q$ output map. C-discriminators containing only 1 C-RAM ($K = 1$) are considered.

In the following, both uni-polar and bipolar notations are used. Latin characters are used for the uni-polar notation and the corresponding Greek characters for the bipolar notation. For instance, if y represents a variable whose value is expressed in the uni-polar notation (i.e. $y \in \{0,1\}$), then v is used to represent the corresponding variable, whose value is expressed in the bipolar notation (i.e. $v \in \{-1,1\}$). The conversion formula between the two notations is:

$$v = 2y - 1.$$

The values stored in a C-RAM can be viewed as 'masses' in an n -dimensional binary space. Let m_i be the value stored, in a C-RAM, at the memory location addressed by input vector

$$\mathbf{x} = (x_{n-1}, x_{n-2}, \dots, x_1, x_0),$$

with

$$i = \sum_{k=0}^{n-1} x_k \cdot 2^k$$

A corresponding mass m_i would therefore be located at co-ordinates

$$\xi^i = (\xi_{n-1}^i \xi_{n-2}^i \cdots \xi_j^i \cdots \xi_0^i)$$

in an n -dimensional binary space. A centre of gravity γ can be calculated, which represents the contents of a C-RAM and is defined as a vector

$$\gamma = (\gamma_{n-1} \gamma_{n-2} \cdots \gamma_j \cdots \gamma_0),$$

with

$$\gamma_j = \frac{1}{M} \sum_{i=0}^{2^n-1} m_i \cdot \xi_j^i \quad (5.10)$$

and

$$M = \sum_{i=0}^{2^n-1} m_i, \quad (5.11)$$

γ_j being a real value such that $\gamma_j \in [-1, 1]$. For example, in the case of a 2-input C-RAM, it holds that:

$$\gamma_0 = \frac{\sum_{i=0}^3 m_i \cdot \xi_0^i}{\sum_{i=0}^3 m_i} = \frac{-m_0 - m_1 + m_2 + m_3}{m_0 + m_1 + m_2 + m_3} \quad (5.12)$$

and

$$\gamma_1 = \frac{\sum_{i=0}^3 m_i \cdot \xi_1^i}{\sum_{i=0}^3 m_i} = \frac{-m_0 + m_1 - m_2 + m_3}{m_0 + m_1 + m_2 + m_3}. \quad (5.13)$$

Figure 5.13 shows, in a 2-dimensional space, the 4 masses corresponding to the 4 memory locations of a 2-input C-RAM, together with the location of the gravity centres

corresponding to characteristic C-RAM contents.

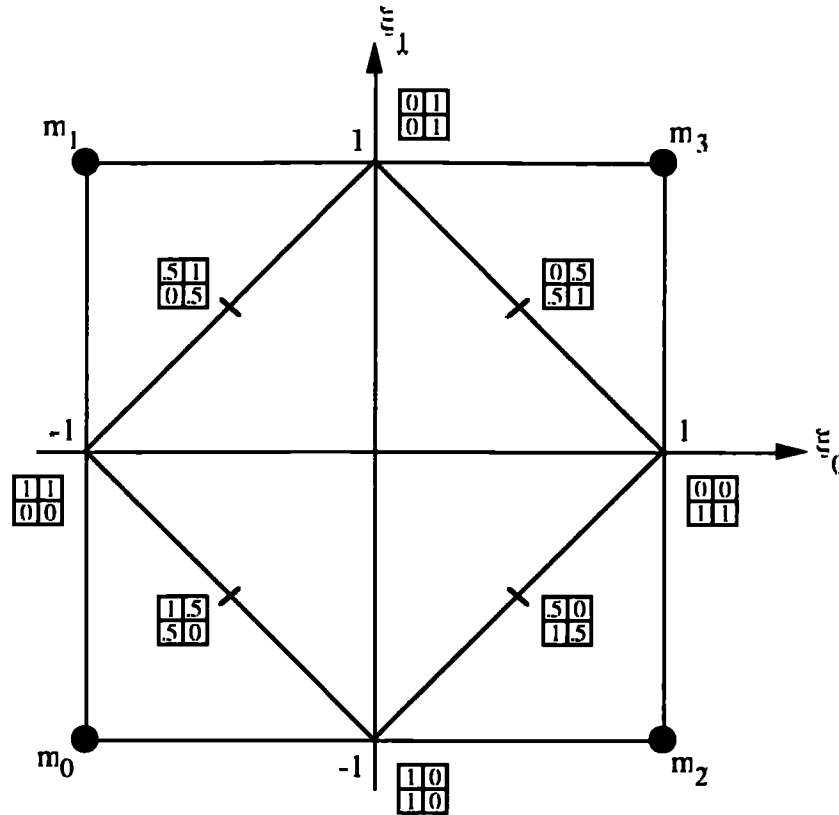


Fig. 5.13: Representation, in a 2-dimensional space, of the masses corresponding to the contents of a 2-input C-RAM. The locations of the gravity centres corresponding to characteristic C-RAM contents are shown.

The co-ordinates (γ_1, γ_0) of the gravity centre of a C-RAM are now used as a pseudo-weight vector. The output f of a C-RAM is first expressed as a polynomial function of its inputs (e.g., [Ull73] or [Gur89]):

$$f = m_0(1 - x_0)(1 - x_1) + m_1(1 - x_0)x_1 + m_2x_0(1 - x_1) + m_3x_0x_1$$

or, using a bipolar notation,

$$f = m_0 \frac{(1 - \xi_0)}{2} \frac{(1 - \xi_1)}{2} + m_1 \frac{(1 - \xi_0)}{2} \frac{(1 + \xi_1)}{2} + m_2 \frac{(1 + \xi_0)}{2} \frac{(1 - \xi_1)}{2} + m_3 \frac{(1 + \xi_0)}{2} \frac{(1 + \xi_1)}{2}. \quad (5.14)$$

Using $m_i(t)$ to denote the value of m_i at time step t and following the training algorithm introduced in Section 5.3.3, the following relations between memory variables can be

written:

$$m_0(t) - m_0(0) = -(m_3(t) - m_3(0)) \quad (5.15)$$

and

$$m_1(t) - m_1(0) = -(m_2(t) - m_2(0)). \quad (5.16)$$

Equations (5.15) and (5.16) are a consequence of the fact that, during training, the values m_0 and m_1 are always updated with a quantity exactly opposite to that with which the values m_3 and m_2 are updated, respectively. Given the initial memory contents $m_i(0) = 1/2$ and dropping the dependency in t , (5.15) and (5.16) become

$$m_3 = 1 - m_0 \quad (5.17)$$

and

$$m_2 = 1 - m_1, \quad (5.18)$$

Substituting (5.17) and (5.18) into (5.14) yields

$$\begin{aligned} 4f &= m_0[(1 - \xi_0)(1 - \xi_1) - (1 + \xi_0)(1 + \xi_1)] + m_1[(1 - \xi_0)(1 + \xi_1) - (1 + \xi_0)(1 - \xi_1)] \\ &\quad + (1 + \xi_0)(1 + \xi_1) + (1 + \xi_0)(1 - \xi_1) \\ &= -2m_0(\xi_0 + \xi_1) - 2m_1(\xi_0 - \xi_1) + 2(1 + \xi_0) \\ &= \xi_0(-2m_0 - 2m_1) + \xi_1(-2m_0 + 2m_1) + 2(1 + \xi_0). \end{aligned} \quad (5.19)$$

Finally, using (5.10) to (5.13), expression (5.19) becomes:

$$4f = \xi_0 \cdot M \cdot \gamma_0 + \xi_1 \cdot M \cdot \gamma_1 + 2$$

or

$$\phi = \xi_0 \cdot w_0 + \xi_1 \cdot w_1, \quad (5.20)$$

with

$$w_i = M \cdot \gamma_i$$

and where ϕ denotes the output of the C-RAM expressed in bipolar notation.

It is possible to show that, in the general case of n inputs per C-RAM, (5.20) takes a form similar to the activation function of a higher order neuron, but in which the terms corresponding to the product of an even number of variables are eliminated:

$$\phi = \sum_i \xi_i \cdot w_i + \sum_{i,j,k} \xi_i \cdot \xi_j \cdot \xi_k \cdot w_{ijk} + \dots$$

A C-discriminator network with parameters $Q = 24$, $K = 1$ and $n = 2$ is considered. The input patterns are randomly drawn from a uniform distribution of input values, that is, at each training step, the input to the network is randomly chosen amongst the four possible input patterns $\{0,0\}$, $\{0,1\}$, $\{1,0\}$ and $\{1,1\}$. In this experiment, the excitation and inhibition radii were chosen to be equal to 6 and 11, respectively. Figure 5.14 shows the network's output map, after about 10,000 training steps, in response to one of the input patterns, say $\{1,1\}$. Similar output maps are obtained for the other 3 input patterns, but with the dark and light areas of the map occupying one of the other possible edges of the map.

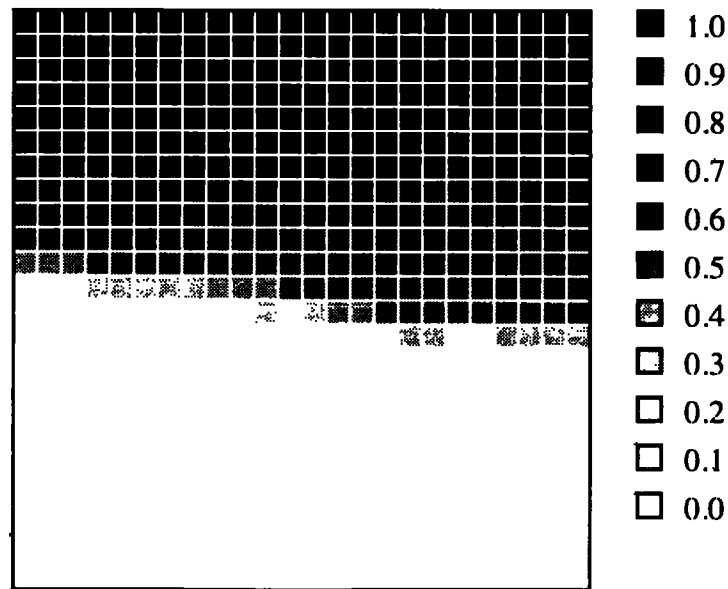


Fig. 14: The network's output map, after about 10,000 training steps, in response to one of the input patterns, say $\{1,1\}$.

A graphical representation of the gravity centres of the network's C-RAMs, after about 10,000 training steps, is shown in Figure 5.15. The spatial arrangement of the neurons is represented by connecting together the centres of gravity of any two spatially adjacent C-RAMs. The main conclusion is that the C-RAMs memory contents has self-organised in order to form an internal representation of the uniform input distribution.

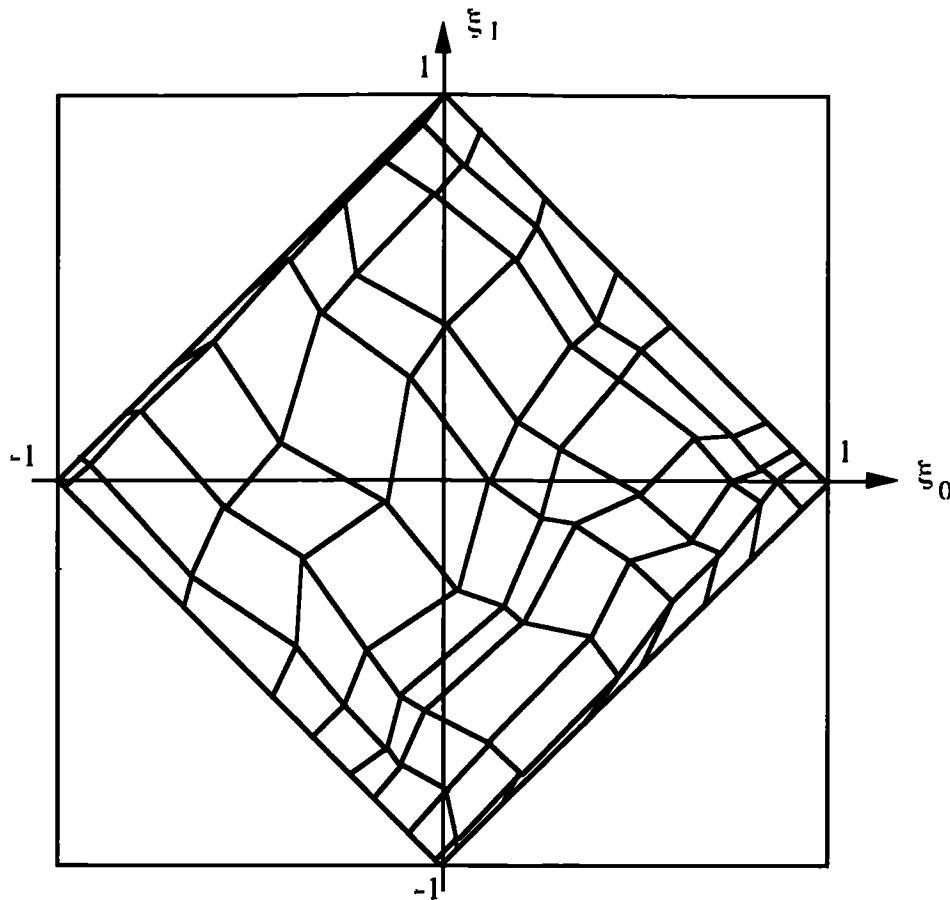


Fig. 5.15: Graphical representation of the gravity centres of the network's C-RAMs, after about 10,000 training steps. The spatial arrangement of the neurons is represented by connecting together the centres of gravity of any two spatially adjacent C-RAMs.

5.5. Comparison with other weightless models

The network and learning algorithm presented in the previous sections were first published in [Nto90] and constitute an *original* contribution to this thesis. Other weightless networks producing self-organisation through some unsupervised algorithm have been proposed (e.g., [AllJM89], [KerS90], [TamS92] or [RicS93]). These were either developed independently, at about the same time as the present work ([AllJM89] and [KerS90]), or subsequently and as a continuation of previous work ([TamS92] and [RicS93]).

In the sequel, the C-discriminator is compared to a weightless self-organising network proposed by Allinson [AllJM89] (also in [AllBJ89], [AllJ89] or [AllJ92]). Similarities and differences with the C-discriminator model are emphasised.

In [AllJM89], a technique is described for realising self-organising feature maps, which "exploits the properties of $\{0,1\}^n$ space" [AllBJ89]. Basically, the network, referred to here as the Allinson network, takes the form of a discriminator-based Kohonen-type network.

The memory locations are only allowed to hold values 0 or 1.

As in the C-discriminator model, the similarity, or *distance metric*, between input patterns and 'prototype' patterns (implicitly defined by the discriminator's memory contents) is measured by the discriminator's response. This similarity is related to the Hamming distance between the input patterns and the prototype patterns. In [AllJM89], [AllBJ89], [AllJ89], and [AllJ92], a insightful interpretation is given of a discriminator's response. This response is interpreted as the number of the times that the input pattern is labelled in the $\{0,1\}^n$ sub-spaces formed by the discriminator's RAMs.

For completion, it is worth mentioning that in the Allinson network, the calculation of the output map is simplified by using a coding scheme that allows the grouping of adjacent units. This is justified by the observation that, in a topologically organised network, adjacent units share similar responses. If the desired output map has dimensions $(Q \times Q)$, then instead of having a network of Q^2 discriminators, only $2 \cdot Q$ discriminators are used: Q discriminators code for the 'x-projection' of the output map and, similarly, Q discriminators code for the 'y-projection'. Therefore, if notations similar to (5.1) are used, the output corresponding to a position (i, j) on the output map can be expressed as:

$$r_{xy} = r_x \cdot r_y = \frac{1}{K^2} \left\{ \sum_{k=0}^K [i_1 \cdots i_n]_k^x \right\} \cdot \left\{ \sum_{k=0}^K [i_1 \cdots i_n]_k^y \right\}.$$

The saturation of the memory content, which occurs in N-tuple systems not endowed with some mechanism for controlling the cluster size, is overcome by maintaining a *fixed* fraction of memory locations set to 1. These locations are initially randomly distributed throughout the RAMs. Then, as training proceeds, memory locations that contained 0 are set to 1, using an updating procedure typical in a Kohonen network and involving the definition of an excitation region about a 'firing' neuron. Each time a memory location is set to 1, another memory location, that was set to 1 in the considered discriminator, is set to 0. The memory location to be cleared is chosen such that it has the same 'norm' (number of 1's in its address N-tuple) as the location that was set to 1. This procedure guarantees a stable learning.

In the C-discriminator network, it is the spreading function which guarantees that there will be no saturation in the system. In fact, the symbol redistribution [AllBJ89] taking place in the Allinson network can be viewed as a simplified form of the spreading function described in this chapter. Indeed, the location addressed by the input pattern in a discriminator in the excitation region, spreads its value, using a negative 'weight', to another location.

In [TamS92], as a continuation of the work described in this Chapter, a simplified spreading function is used, similar to Allinson's, where the location to be cleared is the one 'opposite' to the addressed location, that is, the one at maximum Hamming distance from

the addressed location H . Using the same representation as in Figure 5.4, this spreading function can then be represented as in Figure 5.16.

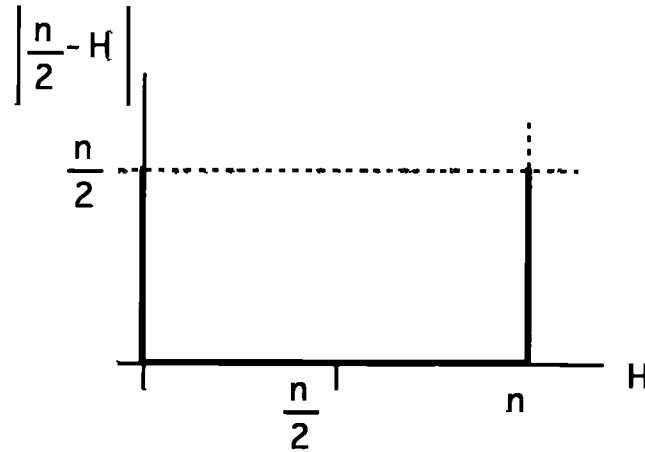


Fig. 5.16: Simplified spreading function, in which the addressed content in a RAM, is only spread to the opposite memory location.

In [KerS90], a self-organising discriminator-based system is described, in which the node functions, set up during training, depend on the 'strength' of the different features present within the training data. The operation of the network, instead of being of the Kohonen type, follows a scheme similar to Grossberg's Adaptive Resonance Theory (ART) (e.g., [Was89], [CarG89] or [CarG90]). The contents of the memory locations is allowed to take an integer value in some range, typically $[-15, 15]$. The output of a node takes a binary value 1 or 0, depending on the sign of the contents of the addressed memory location in the node. Again, the training of the discriminator RAMs makes use of some form of spreading function. The contents of the addressed location within a node is increased by a value ϵ and the contents of *all* other locations, not addressed by the input pattern, is decreased by a value δ , ϵ and δ being integer values in the range $[1, 100[$. By varying the values assigned to ϵ and δ , different 'weights' are attached to the appearance or non-appearance of n -tuple features in the training set. This is equivalent to setting the memory location addressed by some n -tuple only if the corresponding input sub-pattern is present in at least a fraction $\frac{\delta}{\epsilon + \delta}$ of the training data [KerS90].

In [RicS93], a self-organising weightless neural network is presented, which does not make use of an *explicit* spreading procedure. Instead, each iteration of the winner-take-all learning algorithm involves a *batch phase*, during which all the training patterns are presented to the network. This phase allows the calculation of a normalisation factor in order to ensure a zero total update quantity for each RAM. In that way, the mean of the node response is forced to zero. Moreover, a site which is addressed by every pattern will

contain zero after one pass of the data set and will not contribute to the output of the node [RicS93]. In fact, in that system, the spreading of information between memory location contents is implicit in the batch calculation of a common quantity between the memory locations.

5.6. Conclusions

This Chapter was concerned with unsupervised learning in WANNs. A self-organising WANN similar in structure and operation to a Kohonen network has been presented. The C-discriminator node was introduced and it was shown that such a model can be used in a network governed by an unsupervised training procedure. Simulations have shown that the C-discriminator network is able to form a topologically ordered map of input data, where responses to similar patterns are clustered in certain regions of the output map. It was shown that the output responses of the neurons can be predicted on the basis of the overlap areas between training patterns.

The main contribution of this chapter was the introduction of a new training algorithm used to train the C-discriminator network, using a spreading mechanism affecting memory locations not addressed by the training patterns. During the training procedure, the information, concerning how a neuron should respond to new patterns, is spread to each memory location of the C-discriminator's C-RAMs. A C-discriminator situated in the excitation region about the firing neuron will become more susceptible to fire when the same or similar input pattern is presented again. Moreover, as a consequence of the above spreading procedure, this C-discriminator will become less susceptible to fire if an input pattern, very different from the one with which the training phase was performed, is presented to the network. Inverse behaviour is observed for a C-discriminator situated in the inhibition region about the firing neuron. An additional advantage of the spreading algorithm is that the problem of node saturation, which is a characteristic drawback of n -tuple systems (e.g., [BleB59], [Ste62], [Ull73] or [AleTB84]) has been eliminated in the C-discriminator network.

This Chapter was not concerned with implementation issues. In that respect, it should be noted that the C-discriminator training algorithm, as exposed in Section 5.3.3, is not straightforward to implement in hardware. This is due to the continuous values of the memory contents and the spreading of information to each memory location of the nodes at each step of the training algorithm. However, approximations are possible. The memory contents can be restricted to a range of discrete values in the interval $[0,1]$. Furthermore, the spreading function of the training algorithm can be simplified, such as represented in Figure 5.16. Finally, the training of the network can all together be performed by software (see discussion in Section 3.7.2) and the trained network be subsequently implemented in hardware, using existing state-of-the-art random access memory techniques.

CHAPTER VI. WEIGHTLESS AUTO-ASSOCIATIVE MEMORIES

6.1. Introduction

A cluster of interconnected formal neurons has, as an emergent property, the ability to enter stable firing patterns, when stimulated by the presentation of noisy versions of these patterns [Hop82]. The General Neural Unit network (GNU) [Ale90a], reviewed in Section 2.5.3.3, is further studied for its ability to perform auto-association. Hence, the GNU is operated with the number of external inputs N per node being equal to 0 and the number of feedback connections F to a node being different from 0. Furthermore, if a weakly-interconnected network is considered, the value of F represents some small fraction of the total number K of system outputs.

The nodes of a GNU are GRAMs, reviewed in Section 2.3.7. GRAMs operate in three phases. The training phase, similar to the write mode in a RAM, is the operation by which the GRAM records addresses with their required response. The training phase is followed by a spreading phase. Finally, during the use phase, the node makes use of its stored information to produce an output.

The probabilities of disruptions between training patterns in the GNU are calculated and conclusions are drawn on the storage capacity of the network. The method of investigation is purely combinatorial, consisting of counting the number of disruptions between patterns occurring, on average, in the nodes of the GNU. The calculation does not take into account the retrieval characteristics of the GNU, i.e. whether the network converges towards the nearest stored pattern or drifts away in state space. The retrieval process in the GNU network will be examined in Chapter 7.

Finally, a method for improving the immunity of the GNU network to disruptions is also proposed, which uses a network based on the GNU but with discriminator nodes instead of GRAMs.

6.2. Probability of disruption and storage capacity

6.2.1. Assumptions

The assumptions concerning the parameters the considered GNU network have already been stated above, namely, $N = 0$, $F \neq 0$ and $F \ll K$.

The operation of the network is assumed to be synchronous, with all the GRAMs outputting the value addressed by their input F -tuple at the same time.

The question arises of whether it is desirable or not, to allow self feed-back connections to the nodes of the GNU network. In [ZhaS91], self-connection is imposed on all nodes, as a means of eliminating disruptions between patterns. However, this method yields an artificial partitioning of the node memory which leads to the creation of numerous spurious states or false minima. In the following, it is assumed that no self-feedback occurs at the nodes of the GNU. If the input mapping is random, there exists a non-zero probability of self-feedback. The probability P_{self} of self feed-back at a node can be expressed as

$$P_{self} = 1 - \left(1 - \frac{1}{K}\right)^F$$

However, if a weakly interconnected GNU is assumed, i.e. $F \ll K$, then the probability of self-feedback becomes negligible:

$$\begin{aligned} P_{self} &= 1 - \left(1 - \frac{1}{K}\right)^F = 1 - \left(1 - \frac{F}{K} + \dots\right) \\ &\approx \frac{F}{K} \\ &\approx 0. \end{aligned}$$

6.2.2. Probability of disruption of equally and maximally distant patterns

A set L of S training patterns stored in the GNU network,

$$L = \{T_1, T_2, \dots, T_S\},$$

is considered. The patterns are chosen such that the overlap between any two stored patterns is equal and minimal. This assumption seems to be general enough and has the advantage that it will enable an exact calculation of the probability of disruption between patterns. In Section 6.2.2, the high correlation between such patterns is discussed. Thus, the overlap ω_{ij} between patterns T_i and T_j can be expressed as

$$\omega_{ij} = 1 - \frac{2}{S}, \quad (6.1)$$

These patterns can be formed in the following way. The pattern space is divided into S parts containing the same number $\frac{K}{S}$ of pixels each. Any one pattern is formed by

choosing one such part at random and assigning the value 1, or black colour, to its pixels. The value 0, or the white colour, is assigned to the pixels of the other $S - 1$ parts. One possible graphical arrangement of such patterns, which will help understand the subsequent developments, might be to represent an input pattern as divided into S sections $s_1, s_2, \dots, s_S$ of equal area. Section s_j , corresponding to pattern T_j , contains 1-values and all other sections contain 0-values. This arrangement is shown in Figure 6.1.

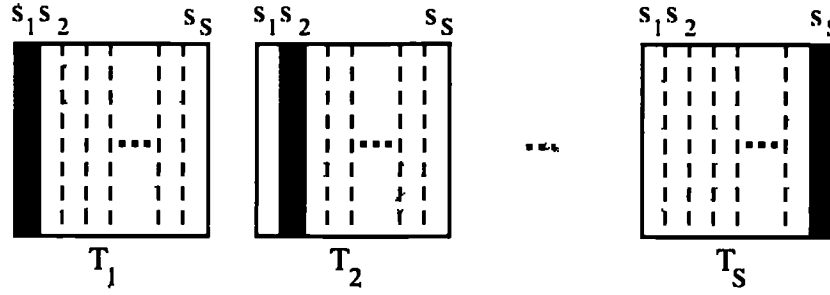


Fig. 6.1: Graphical representation of one possible arrangement of S equally and maximally distant training patterns.

During the storage process, contradictions occur when new stored patterns cause old ones to no longer be stable patterns in the network. In the following, the probability of disruption of pattern T_1 by patterns $T_2, \dots, T_S$ is calculated.

First is considered the probability of pattern T_1 being disrupted by pattern T_2 . This occurs when the input sub-patterns to a particular node are identical for patterns T_1 and T_2 , while the required outputs are different. In other words, a contradiction occurs because a value 1 and a value 0 have to be stored at the same memory location. The probability of this occurring is

$$Pr(T_1 \xleftarrow{\text{disrupt}} T_2) = \frac{2}{S} \cdot \left(1 - \frac{2}{S}\right)^F, \quad (6.2)$$

In (6.2), the factor $\frac{2}{S}$ represents the probability of having different outputs for T_1 and T_2 , while the factor $\left(1 - \frac{2}{S}\right)^F$ is the probability that all inputs to the node are the same for T_1 and T_2 .

Further, the probability that pattern T_1 is disrupted by at least one of the other patterns $T_2, \dots, T_S$, is considered. The following theorem can then be stated:

Theorem 6.1: In a weakly interconnected auto-associative GNU with F -input nodes and storing S equally and maximally distant patterns, the probability $P_d(S, F)$ of disruption of any one pattern, in a particular node, due to the other $S-1$ patterns is given by:

$$P_d(S, F) = \Pr(T_i \xleftarrow{\text{disrupt}} \{T_1, T_2, \dots, T_S\} \setminus T_i)$$

$$= \frac{1}{S} \cdot \left\{ 1 - \left[1 - \left(1 - \frac{2}{S} \right)^F \right]^{S-1} \right\} + \frac{S-1}{S} \cdot \left(1 - \frac{2}{S} \right)^F \quad (6.3)$$

Proof: Let y_i denote the value required at the output of a particular node, when pattern T_i is present at the output terminal of the GNU, and $\mathbf{x}_j$ the vector of feed-back inputs to that same node, for the same pattern T_j . All different cases of contradictions need to be considered.

First, the case when $y_i \neq y_j$, for any j such that $j \neq i$, is considered. Taking into account (6.1), this case has a probability $\frac{1}{S}$. For the inputs, the probability that $\mathbf{x}_i$ is equal to at least another input $\mathbf{x}_j$, with $j \neq i$, needs to be considered. This probability can be expressed as

$$1 - \left[1 - \left(1 - \frac{2}{S} \right)^F \right]^{S-1}.$$

This provides the first term of the overall probability of disruption:

$$\frac{1}{S} \cdot \left\{ 1 - \left[1 - \left(1 - \frac{2}{S} \right)^F \right]^{S-1} \right\}. \quad (6.4)$$

Then, the $S-1$ cases, with

$$k = 2, \dots, S,$$

are considered, for each of which $y_i \neq y_k$ and $y_i = y_j$, for any j such that $j \neq k$ and $j \neq i$. These $S-1$ cases have each a probability of $\frac{1}{S}$. For the inputs, the probability that $\mathbf{x}_i = \mathbf{x}_k$ needs to be considered. This is given by

$$\left(1 - \frac{2}{S} \right)^F.$$

The probability that $x_i = x_j$, with $j \neq i$, need not be considered, since $y_i = y_j$ and therefore no contradiction can occur for that case. This yields the second term in the overall probability of disruption:

$$\frac{S-1}{S} \cdot \left(1 - \frac{2}{S}\right)^F \quad (6.5)$$

Finally, summing the contributions (6.4) and (6.5), the overall probability of disruption $P_d(S, F)$ is obtained:

$$P_d(S, F) = \frac{1}{S} \cdot \left\{ 1 - \left[1 - \left(1 - \frac{2}{S}\right)^F \right]^{S-1} \right\} + \frac{S-1}{S} \cdot \left(1 - \frac{2}{S}\right)^F.$$

This concludes the proof of Theorem 6.1.

The following corollaries can then be formulated:

Corollary 6.1: *The proportion of patterns that are disrupted, on average, in a particular node of a weakly interconnected auto-associative GNU network, with F -input nodes and storing S equally and maximally distant patterns, is given by $P_d(S, F)$.*

Proof: From theorem 6.1, $P_d(S, F)$ is the probability of disruption of a stored pattern, in a particular node, due to all the other stored patterns. Therefore, on average, there is a proportion $P_d(S, F)$ of disrupted patterns. This results from the fact that the mean value of a binomial distribution with parameters S and P_d is given by $S \cdot P_d$.

Corollary 6.2: *In a weakly interconnected auto-associative GNU network, with F -input nodes and storing S equally and maximally distant patterns, the proportion of nodes, on average, in which a particular training pattern is disrupted is given by $P_d(S, F)$.*

Proof: The proof follows from considering theorem 6.1 and the fact that the mean value of a binomial distribution with parameters K and P_d is given by $K \cdot P_d$.

It is interesting to consider the following limit cases:

Corollary 6.3: *A weakly interconnected auto-associative GNU network, with F -input nodes and storing S equally and maximally distant patterns, can store two opposite*

patterns without disruption.

Proof: If $S = 2$, then $P_d(S, F) = P_d(2, F) = 0$. It is indeed obvious that any GNU can store two opposite patterns without disruption, because at each node, the patterns will address a different memory location.

Corollary 6.4: *When an asymptotically large number of equally and maximally distant patterns is stored in a weakly interconnected auto-associative GNU network, they all get disrupted.*

Proof: For large S , the probability of disruption tends to 1;

$$\lim_{S \rightarrow \infty} P_d(S, F) = 1,$$

Corollary 6.5: *When, in a weakly interconnected auto-associative GNU, the number S of stored patterns, assumed to be equally and maximally distant, and the number F of inputs per node become asymptotically large, then $\frac{S}{e^2}$ patterns are disrupted, on average, in any node of the network.*

Proof: If S and F are held at the same value in (6.3), the probability of disruption, in the limit $S = F \rightarrow \infty$, becomes:

$$\begin{aligned} \lim_{S=F \rightarrow \infty} P_d(S, F) &= \lim_{S \rightarrow \infty} \frac{1}{S} \cdot \left\{ 1 - \left[1 - \left(1 - \frac{2}{S} \right)^S \right]^{S-1} \right\} + \lim_{S \rightarrow \infty} \frac{S-1}{S} \cdot \left(1 - \frac{2}{S} \right)^S \\ &= \lim_{S \rightarrow \infty} \left(1 - \frac{2}{S} \right)^S \\ &= \lim_{\varepsilon \rightarrow 0} (1 - 2\varepsilon)^{\frac{1}{\varepsilon}} \\ &= \frac{1}{e^2}. \end{aligned}$$

Then, taking into account Corollary 6.1, the result of Corollary 6.5 follows. This concludes the proof of Corollary 6.5.

Thus, in the above case, a fraction $\frac{1}{e^2}$, or 13.53 %, of the total number S of patterns is disrupted in any node of the network. From Corollary 6.5, the next asymptotic result can be formulated:

Corollary 6.6: *When the number S of stored patterns in a weakly interconnected auto-associative GNU network of K nodes, storing equally and maximally distant patterns, and the number F of inputs per node are asymptotically large, a number $\frac{K}{e^2}$ of nodes exist, on average, in which a particular training pattern is disrupted.*

Proof: The result follows immediately from Corollaries 6.2 and 6.5.

The previous result shows that the GNU network, under the conditions specified by Corollary 6.6, can store about the same number S of patterns as the number F of inputs per node, with a small error due to memory locations containing u values being addressed and therefore outputting at random. This disruption error means that, provided a stored pattern is stable, a number $\frac{K}{2e^2}$ of its pixels would, on average, have incorrect values at the terminals of the GNU, resulting in a noisy version of the original stored pattern.

Although Corollary 6.6 was formulated in the asymptotic case where both variables F and S tend towards infinity, the result holds approximately for finite values of the variables. For example, an auto-associative GNU with a number of feed-back inputs per node $F=8$, could store 8 equally and maximally distant patterns with a disruption probability of 0.145, which is a good approximation of the theoretical value 0.135. The above storage capacity result can now be stated in a formal way:

Theorem 6.2: *The number S of equally and maximally distant patterns that can be stored in a weakly interconnected auto-associative GNU network is approximately equal to the number F of inputs per node.*

Proof: Considering the result of Corollary 6.5, it holds that

$$P_d(S, F) \approx e^{-2}, \quad (6.6)$$

Furthermore, when S and F become large, the second term of (6.3), also given by (6.5), becomes much more important than the first term, also given by (6.4), and the following relation holds:

$$P_d(S, F) \approx \frac{S-1}{S} \cdot \left(1 - \frac{2}{S}\right)^F. \quad (6.7)$$

Combining (6.6) and (6.7) and taking the logarithm on each side yields

$$-2 \approx \log\left(\frac{S-1}{S}\right) + F \cdot \log\left(1 - \frac{2}{S}\right). \quad (6.8)$$

The first term of the right member of (6.8) is negligible and the first term of the development

$$\log(1-x) \approx -x - \frac{x^2}{2} - \frac{x^3}{3} - \dots, \quad (6.9)$$

holding for

$$-1 < x < 1,$$

can be used on the second logarithm in (6.8). This yields

$$-2 \approx F \cdot \left(-\frac{2}{S}\right)$$

and finally

$$S \approx F,$$

which concludes the proof of Theorem 6.2.

The main conclusion is that the proportion of disrupted patterns, and therefore the storage capacity of the network, depend primarily on the number F of inputs to a node. A similar result is reported in [Ale90c]. Also, this storage capacity result, calculated as a count of potentially stable patterns in a GNU, is not a sufficient indicator of whether the patterns are stable or metastable. Therefore the retrieval properties of the system will need to be examined. This is carried out in Chapter 7.

6.2.3. Experimental verification of Corollary 6.5

In order to verify experimentally the validity of the theoretical result expressed by Corollary 6.5, a simulation is run which counts the proportion of patterns which are disrupted at a particular node, for the case of equally and maximally distant patterns. For example, the case

$$S = F = 100$$

and

$$K = 10,000,$$

having therefore $K \gg F$, is simulated. As there are 10,000 nodes in the network and in order for the GNU input terminal to have the same size as the output terminal, the training patterns considered consist of arrays of 100×100 pixels. In any such training pattern, all the pixels are white except for an area of size 10 by 10 and which consists of black pixels. The black area is chosen to be different for each training pattern. This type of patterns constitute one possible example of equally and maximally distant patterns.

Then, the following procedure, which simulates the storage, in a particular node, of the training patterns, is repeated n times. This procedure consists of:

1. choosing F random pixels for the inputs to the node and 1 random pixel for the output of the node;
2. mapping the pixels values, for each one of the 100 training patterns, to the inputs and output of the node;
3. determining whether there is at least one contradiction, that is, the node is required to output a different value for the same input address. If, during an iteration, at least one contradiction is found, a count variable c is incremented by one.

The ratio $\frac{c}{n}$ is observed to converge towards a value of about 0.135, which corresponds to the theoretical result expressed by Corollary 6.5.

6.2.4. Equally and maximally distant patterns

In section 6.2.2, equally and maximally distant, or minimally overlapping, training patterns were used for the calculations of disruption probabilities. It is interesting to note that the correlation between those patterns varies with their number S .

The correlation between patterns T_1 and T_2 is defined as:

$$\rho_{12} = \frac{1}{m} \sum_{i=1}^m \tau_1^i \cdot \tau_2^i,$$

in which m is the total number of pixels in the patterns, τ_1^i and τ_2^i being the pixel values of the two patterns, in polarised notation. When $\rho_{12} = -1$, the two patterns are opposed, whereas when $\rho_{12} = 1$, the two patterns are identical. A value $\rho_{12} = 0$ means that the two patterns are uncorrelated. The relation between overlap ω and correlation ρ is given by:

$$\rho = \omega - (1 - \omega) = 2\omega - 1,$$

Using (6.1), the overlap ρ_{ij} between patterns T_i and T_j can be expressed as:

$$\rho_{ij} = 2\omega_{ij} - 1 = 1 - \frac{4}{S},$$

with $S \geq 2$. This is represented graphically in Figure 6.2.

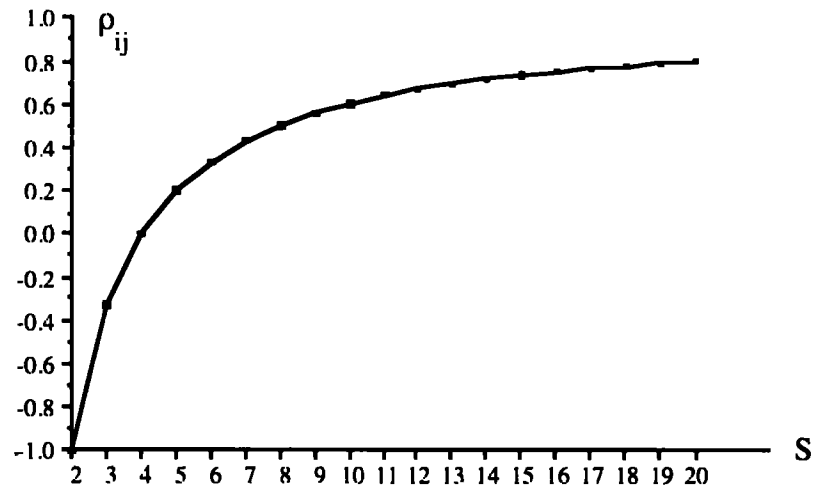


Fig. 6.2: Graph showing the correlation between any two stored patterns as a function of the total number S of patterns. The patterns are assumed to be equally and maximally distant from each other.

In the case of 2 training patterns ($S = 2$), these are opposite patterns and therefore maximally distant from each other. When $S = 4$, the correlation becomes 0. It then continues increasing with the number S of patterns to reach an asymptotic value of 1. The asymptotic result of Corollary 6.5 would seem, therefore, to be a consequence of the high similarity between patterns when S is large. However, as shown before, the storage result expressed in Theorem 6.2 holds also for small values of S .

It is interesting to calculate the probability of disruption in the case of uncorrelated or random patterns.

6.2.5. Probability of disruption of uncorrelated patterns

The patterns considered in this section are uncorrelated, that is

$$\forall i, j \in \{1, 2, \dots, S\}, i \neq j : \rho_{ij} = 0.$$

Therefore the average overlap between patterns becomes:

$$\omega_{ij} = \frac{1 + \rho_{ij}}{2} = \frac{1}{2} \quad (6.10)$$

The probability $P_d(S, F)$ of disruption, at a particular node of a GNU, of a pattern T_i by one or more of the other $S - 1$ patterns is considered. The following theorem can be stated:

Theorem 6.3: *In a weakly interconnected auto-associative GNU with F -input nodes and storing S random patterns, the probability $P_d(S, F)$ of disruption of any one pattern, in a particular node, due to the other $S - 1$ patterns is given by:*

$$\begin{aligned} P_d(S, F) &= \Pr(T_i \xleftarrow{\text{disrupt}} \{T_1, T_2, \dots, T_S\} \setminus T_i) \\ &= 1 - \left(1 - 2^{-(F+1)}\right)^{S-1}. \end{aligned} \quad (6.11)$$

Proof: The reasoning is similar to the one followed in Section 6.2.2. Again, y_i denotes the value required at the output of a particular node, when pattern T_i is present at the output terminal of the GNU, and $\mathbf{x}_i$ the vector of feed-back inputs to that same node, for the same pattern T_i . The probability of disruption is calculated for pattern T_1 , i.e. $i = 1$, without loss of generality.

The $S - 1$ patterns $T_2, \dots, T_S$ are divided into 2 sets, the first set containing $q - 1$ patterns T_j such that

$$y_1 \neq y_j, \text{ with } j \in \{2, 3, \dots, q\}$$

and the second $S - q$ patterns such that

$$y_1 = y_j, \text{ with } j \in \{q + 1, q + 2, \dots, S\}.$$

There are $\binom{S-1}{q-1}$ ways of choosing $q - 1$ patterns amongst $S - 1$ patterns and the probability of each occurring for a pattern T_j , such that $y_1 \neq y_j$, can be expressed as:

$$(1 - \omega_{1j})^{q-1} \cdot \omega_{1j}^{S-q} \quad (6.12)$$

Taking (6.10) into account, (6.12) becomes:

$$\left(\frac{1}{2}\right)^{q-1} \cdot \left(\frac{1}{2}\right)^{S-q} = 2^{1-S}, \quad (6.13)$$

For the inputs to the considered node, the probability that the input vector $\mathbf{x}_i$, corresponding to pattern $T_{i\neq}$ is identical to at least another of the $q-1$ patterns T_k for which $y_i \neq y_j$, is

$$1 - \left[1 - \left(\frac{1}{2}\right)^F\right]^{q-1} \quad (6.14)$$

Finally, summing the product of (6.13) by (6.14) over the $S-1$ cases, corresponding to the values $q \in \{2, 3, \dots, S\}$, an expression for the probability of disruption $P_d(S, F)$ is derived:

$$\begin{aligned} P_d(S, F) &= \sum_{q=2}^S \binom{S-1}{q-1} \cdot 2^{1-S} \cdot \left[1 - (1 - 2^{-F})^{q-1}\right] \\ &= 2^{1-S} \cdot \left\{ \sum_{q=2}^S \binom{S-1}{q-1} \right\} - 2^{1-S} \cdot \left\{ \sum_{q=2}^S \binom{S-1}{q-1} \cdot (1 - 2^{-F})^{q-1} \right\} \\ &= 2^{1-S} \cdot \{2^{S-1} - 1\} - 2^{1-S} \cdot \left\{ \sum_{r=1}^{S-1} \binom{S-1}{r} \cdot (1 - 2^{-F})^r \cdot (1)^{S-1-r} \right\} \\ &= 1 - 2^{1-S} - 2^{1-S} \cdot \left\{ (2 - 2^{-F})^{S-1} - 1 \right\}, \end{aligned}$$

Finally,

$$P_d(S, F) = 1 - \left[1 - 2^{-(F+1)}\right]^{S-1}$$

which concludes the proof of Theorem 6.3.

Alternative proof: The probability of disruption of pattern T_i by pattern T_j is given by

$$\Pr(T_i \xrightarrow{\text{disrupt}} T_j) = (1 - \omega_{ij}) \cdot \omega_{ij}^F$$

Taking into account (6.10), it holds that

$$\Pr(T_i \xrightarrow{\text{disrupt}} T_j) = 2^{-(F+1)}$$

and therefore the overall probability of disruption becomes

$$P_d(S, F) = 1 - \prod_{i=1}^S \left[1 - \Pr(T_i \xrightarrow{\text{disrupt}} T_i) \right]$$

and finally

$$P_d(S, F) = 1 - \left[1 - 2^{-(F+1)} \right]^{S-1}$$

which concludes the alternative proof of Theorem 6.3.

Corollaries 6.1, 6.2 and 6.4, stated above for the case of equally and maximally distant patterns, are still valid in the case of random patterns. Corollary 6.3 is only approximately valid in the case of random patterns, since there is a non-zero probability of disruption in the case of 2 random patterns stored in the GNU. In that case, (6.11) becomes

$$P_d(F, 2) = 2^{-(F+1)}.$$

In the case of random patterns, Corollary 6.5 does not hold anymore and is replaced by the following corollary:

Corollary 6.7: *When, in a weakly interconnected auto-associative GNU, the number S of stored patterns, assumed to be random, and the number F of inputs per node become asymptotically large, then no pattern gets disrupted, on average, in any node of the network.*

Proof: If S and F are held at the same value in (6.11), the probability of disruption, in the limit $S = F \rightarrow \infty$, becomes:

$$P_d(S) = \lim_{S \rightarrow \infty} \left(1 - \left[1 - 2^{-(S+1)} \right]^{S-1} \right) = 0.$$

Thus:

$$\lim_{F=S \rightarrow \infty} P_d(F, S) = 0.$$

which concludes the proof of Corollary 6.7.

The previous asymptotic result, which is also approximately valid for finite values of the variables F and S , shows that a weakly interconnected auto-associative GNU can easily store F random patterns without disruption. This means that the maximum number of patterns that the network can store, with a small probability of disruption, is likely to be much higher than the number F of inputs per node. The storage capacity becomes then of the order of the number 2^F of memory locations per node in the GNU. Therefore, the following corollary can be stated:

Corollary 6.8: *When, in a weakly interconnected auto-associative GNU, the number S of stored patterns, assumed to be random, and the number 2^F of memory locations per node become asymptotically large, then $S \cdot \left(1 - \frac{1}{\sqrt{e}}\right)$ patterns are disrupted, on average, in any node of the network.*

Proof: If S is held at the same value as 2^F in (6.11), then the probability of disruption, in the limit $S = 2^F \rightarrow \infty$, becomes:

$$\begin{aligned} \lim_{S=2^F \rightarrow \infty} P_d(S, F) &= \lim_{F \rightarrow \infty} \left[1 - \left(1 - 2^{-(F+1)} \right)^{2^F - 1} \right] \\ &= 1 - \lim_{F \rightarrow \infty} \left(1 - \frac{1}{2} \cdot \frac{1}{2^F} \right)^{2^F} \\ &= 1 - \lim_{\epsilon \rightarrow 0} \left(1 - \frac{\epsilon}{2} \right)^{\frac{1}{\epsilon}} \\ &= 1 - e^{-\frac{1}{2}}. \end{aligned}$$

Then, taking into account Corollary 6.1, which is also valid for random patterns, the result of Corollary 6.8 follows. This concludes the proof of Corollary 6.8.

Thus, in the above case, a fraction $1 - \frac{1}{\sqrt{e}}$, or 39.35%, of the total number S of patterns is disrupted in any node of the network. By taking into account Corollary 6.2, which is still valid for random patterns, it can be said that, in a GNU of K nodes, storing 2^F random patterns, a number $K \cdot \left(1 - \frac{1}{\sqrt{e}}\right)$ of nodes exist, on average, in which a particular training pattern is disrupted. This disruption error seems to be no longer acceptable, as this was the case for the disruption of equally and maximally distant patterns, obtained in Corollary 6.5. Therefore, a storage ratio α is defined as the proportion of the 2^F patterns that a GNU with F -inputs per node could store. It holds that

$$S = \alpha \cdot 2^F, \quad (6.15)$$

with

$$0 < \alpha < 1.$$

Theorem 6.4: A weakly interconnected auto-associative GNU network, with F inputs per node, can store a number $\alpha \cdot 2^F$ of random patterns, where α is a storage ratio which is directly related to the probability $P_d(S, F)$ of disruption between patterns, by the relation:

$$\alpha = 2 \cdot P_d(S, F). \quad (6.16)$$

Proof: If S is held at the same value as $\alpha \cdot 2^F$ in (6.11), then the probability of disruption, in the limit $S = \alpha \cdot 2^F \rightarrow \infty$, becomes:

$$\begin{aligned} P_d &= \lim_{S=\alpha \cdot 2^F \rightarrow \infty} P_d(S, F) = \lim_{F \rightarrow \infty} \left[1 - \left(1 - 2^{-(F+1)} \right)^{\alpha \cdot 2^F - 1} \right] \\ &= 1 - \lim_{\epsilon \rightarrow 0} \left(1 - \frac{\alpha}{2} \cdot \epsilon \right)^{\epsilon} \\ &= 1 - e^{-\frac{\alpha}{2}}. \end{aligned} \quad (6.17)$$

Equation (6.17) can be rewritten as:

$$\log(1 - P_d) = -\frac{\alpha}{2}. \quad (6.18)$$

Finally, replacing the logarithm in (6.18), by the first term of the development (6.9), it holds that:

$$\alpha = 2 \cdot P_d,$$

which concludes the proof of Theorem 6.4.

Alternative Proof: Using (6.11), an expression for the storage capacity of the network is derived:

$$S = 1 + \frac{\log(1 - P_d)}{\log(1 - 2^{-(F+1)})} \quad (6.19)$$

Substituting both logarithms in (6.19) by the first term of the development (6.9), it holds that:

$$\begin{aligned} S &\approx 1 + P_d \cdot 2^{F+1} \\ &\approx P_d \cdot 2^{F+1}. \end{aligned} \quad (6.20)$$

Finally, the substitution of (6.15) into (6.20) yields

$$\alpha = 2 \cdot P_d,$$

which concludes the alternative proof of Theorem 6.4.

Equation (6.16), which shows the direct correspondence between the storage ratio α and the probability of disruption P_d , means that the greater the fraction of the patterns allowed to be disrupted in the GNU nodes, the more patterns can be stored in the network. However, the probability of disruption needs to remain small in order to enable correct retrieval of the stored patterns. For example, considering a weakly interconnected auto-associative GNU with 8 inputs per node, i.e. $F = 8$, and if a probability of disruption of 10% is assumed to be acceptable, then, from (6.15) and (6.16), it can be seen that the network is able to store about 51 random patterns.

It is also interesting to note that, for the case $\alpha = 1$, (6.15) shows that any stored pattern is disrupted, on average, in 50% of the nodes, which can be considered to be a good approximation of the figure of 39% provided by Corollary 6.8.

6.3. Improving the immunity of the GNU network to contradictions

It was shown, in the previous sections, that stored patterns get disrupted in the nodes of a GNU, due to contradictions occurring because a node is required to store opposite values at the same memory location. When such a contradiction occurs, a value u is stored at the memory location in question.

In a standard GNU network, only one GRAM codes for each output of the network. If the output of the GRAM is wrong, then the corresponding network output will also be wrong. In order to improve the immunity of the network to contradictions, a GNU is

considered, in which several GRAMs are allowed to code for each of the outputs of the GNU. Thus a node of the GNU becomes a kind of discriminator of GRAMs, in which the GRAM outputs are summed and thresholded. A threshold of 0.5 is chosen, in such a way that it implements a sort of voting system between several GRAMs.

When a stored pattern, assumed to be un-distorted, addresses the nodes of the GNU, each GRAM either outputs the correct value for that pattern or, in the case of a contradiction between stored patterns, the GRAM outputs the value u . A vote is operated amongst the outputs of the nodes coding for the same GNU output, and 0, 1 or u is decided for the GNU output. This method, consisting of having more than one GRAM coding for a GNU output, increases the probability that at least one of the GRAMs coding for a particular output will output a non- u value.

A number M of GRAMs is assumed to be coding for each output of the GNU. Using the same methodology as in the previous sections, the following two theorems can be stated:

Theorem 6.5: *The number S of equally and maximally distant patterns that can be stored in a weakly interconnected auto-associative GNU network, with F inputs per node and in which a number M of GRAMs code for each GNU output, is given by:*

$$S = M \cdot F. \quad (6.21)$$

Proof: The probability $P_d(S, F)$ of disruption of any one pattern, in a particular GRAM, due to the other $S - 1$ patterns is given by (6.3). Since the input mapping of each GRAM is random and independent of the mapping of the other GRAMs, therefore the probability $P_d(S, F, M)$ of disruption of any pattern in all M GRAMs coding for the same GNU output is given by:

$$P_d(S, F, M) = [P_d(S, F)]^M. \quad (6.22)$$

If S is held at the same value as the product $F \cdot M$, then the probability of disruption $P_d(S, F, M)$, in the limit $S = F \cdot M \rightarrow \infty$, becomes:

$$\begin{aligned} \lim_{S=F \cdot M \rightarrow \infty} P_d(S, F, M) &= \lim_{S \rightarrow \infty} \left[\frac{1}{S} \cdot \left\{ 1 - \left[1 - \left(1 - \frac{2}{S} \right)^{\frac{S}{M}} \right]^{S-1} \right\} + \frac{S-1}{S} \cdot \left(1 - \frac{2}{S} \right)^{\frac{S}{M}} \right]^M \\ &= \left[\lim_{S \rightarrow \infty} \left(1 - \frac{2}{S} \right)^{\frac{S}{M}} \right]^M \end{aligned}$$

$$\begin{aligned}
&= \left[\lim_{S \rightarrow \infty} \left(1 - \frac{2}{M} \cdot \frac{M}{S} \right)^{\frac{S}{M}} \right]^M \\
&= \left[e^{-\frac{2}{M}} \right]^M = e^{-2}.
\end{aligned}$$

This asymptotic result being approximately valid for finite values of the variables, it holds that:

$$P_d(S, F, M) \approx e^{-2}.$$

Moreover, similarly to (6.7), it holds that:

$$P_d(S, F, M) \approx \left[\frac{S-1}{S} \cdot \left(1 - \frac{2}{S} \right)^F \right]^M.$$

Finally, following a similar argument to the one in the proof of Theorem 6.2, the result

$$S \approx F \cdot M$$

is obtained. This concludes the proof of Theorem 6.5.

Theorem 6.6: *A weakly interconnected auto-associative GNU network, with F inputs per node and in which a number M of GRAMs code for each GNU output, can store a number $\alpha \cdot 2^F$ of random patterns, where α is a storage ratio which is directly related to the probability of disruption $P_d(S, F, M)$, by the relation:*

$$\alpha = 2 \cdot \sqrt[M]{P_d(S, F, M)}. \quad (6.23)$$

Proof: The probability $P_d(S, F, M)$ of disruption of any pattern in all M GRAMs coding for the same GNU output is given by (6.22), in which the probability $P_d(S, F)$ of disruption of any one pattern, in a particular GRAM, due to the other $S-1$ patterns, is given by (6.11). If S is held at the same value as $\alpha \cdot 2^F$ in (6.11), then the probability of disruption $P_d(S, F, M)$, in the limit $S = \alpha \cdot 2^F \rightarrow \infty$, becomes:

$$P_d = \lim_{S = \alpha \cdot 2^F \rightarrow \infty} P_d(S, F, M) = \lim_{S = \alpha \cdot 2^F \rightarrow \infty} [P_d(S, F)]^M$$

$$\begin{aligned}
&= \lim_{F \rightarrow \infty} \left[1 - \left(1 - 2^{-(F+1)} \right)^{\alpha \cdot 2^F - 1} \right]^M \\
&= \left\{ \lim_{F \rightarrow \infty} \left[1 - \left(1 - 2^{-(F+1)} \right)^{\alpha \cdot 2^F - 1} \right] \right\}^M \\
&= \left\{ 1 - e^{-\frac{\alpha}{2}} \right\}^M.
\end{aligned} \tag{6.24}$$

Equation (6.24) can be rewritten as:

$$\log \left[1 - (P_d)^{\frac{1}{M}} \right] = -\frac{\alpha}{2}. \tag{6.25}$$

Finally, replacing the logarithm in (6.25), by the first term of the development (6.9), it holds that:

$$\alpha = 2 \cdot \sqrt{P_d},$$

which concludes the proof of Theorem 6.6.

Equation (6.23) shows the direct correspondence between the storage ratio α and the probability of disruption $P_d(S, F, M)$ allowed in the GNU. The use of more than one GRAM to code for a GNU output results in an increased storage capacity. The same example as in Section 6.2.5 is considered, that is, a weakly interconnected auto-associative GNU with $F = 8$ inputs per node. A probability of disruption of 10% is assumed to be acceptable, that is, $P_d(S, F, M) = 0.1$. If 2 GRAMs are used to code for each GNU output, i.e. $M = 2$, then it can be calculated, using (6.23) and (6.15), that the network is able to store about 162 random patterns.

6.4. Conclusions

The ability of a weakly interconnected GNU network to perform auto-association was studied. The disruption probabilities between training patterns in the GNU were calculated and the storage capacity of the network was derived. Two kinds of patterns were considered: first, equally and maximally distant patterns; and then, random patterns. The method of investigation was purely combinatorial, consisting of counting the average number of disruptions between patterns, occurring in the GNU network storing a number S of patterns.

It was proved that, when the number S of equally and maximally distant patterns.

stored in a weakly interconnected auto-associative GNU network of K nodes, and the number F of inputs per node are asymptotically large, a number $\frac{K}{e^2}$ of nodes exist, on average, in the GNU, in which a particular training pattern is disrupted. Therefore, under the above conditions, the GNU network can store about the same number S of patterns as the number F of inputs per node, with a small error due to memory locations containing u values being addressed and therefore outputting at random. This disruption error means that, provided a stored pattern is stable, a number $\frac{K}{2e^2}$ of its pixels will, on average, have incorrect values at the terminals of the GNU, resulting in a noisy version of the original stored pattern.

It was proved that, when, in a weakly interconnected auto-associative GNU, the number S of stored patterns, assumed to be random, and the number 2^F of memory locations per node become asymptotically large, then a fraction $1 - \frac{1}{\sqrt{e}}$, or 39.35%, of the total number S of patterns is disrupted in any node of the network. Therefore, in a GNU of K nodes, storing 2^F random patterns, a number $K \cdot \left(1 - \frac{1}{\sqrt{e}}\right)$ of nodes exist, on average, in which a particular training pattern is disrupted.

It was proved that a weakly interconnected auto-associative GNU network, with F inputs per node, can store a number $\alpha \cdot 2^F$ of random patterns, where α is a storage ratio which is directly related to the probability $P_d(S, F)$ of disruption between patterns, by the relation $\alpha = 2 \cdot P_d(S, F)$. This means that the greater the fraction of the patterns allowed to be disrupted in the GNU nodes, the more patterns can be stored in the network. However, the probability of disruption needs to remain small in order to enable correct retrieval of the stored patterns.

Finally, a method for improving the immunity of the GNU network to disruptions was proposed, which used a network based on the GNU but in which more than one GRAM code for each GNU output bit. It was proved that the number S of equally and maximally distant patterns that can be stored in a weakly interconnected auto-associative GNU network, with F inputs per node and in which a number M of GRAMs code for each GNU output, is given by $S = M \cdot F$. In the case of random patterns, it was proved that the network can store a number $\alpha \cdot 2^F$ of random patterns, where α is a storage ratio which is directly related to the probability of disruption $P_d(S, F, M)$, by the relation $\alpha = 2 \cdot \sqrt[M]{P_d(S, F, M)}$. In conclusion, the use of more than one GRAM to code for a GNU output results in an increased storage capacity of the network.

CHAPTER VII. RETRIEVAL PROCESS IN THE GENERAL NEURAL UNIT NETWORK

7.1. Introduction

In this chapter, the retrieval process in the auto-associative GNU network is studied.

In the first section, the relationships between pattern overlaps are derived and several useful corollaries are formulated, using the principle of inclusion and exclusion. One of the results is that the overlap between a odd number of patterns can be expressed as a linear function of lower order overlaps, whereas no such expression exists in the case of the overlap between an even number of patterns.

A spreading process with full generalisation takes place in the GNU nodes after training. The associated spreading function is derived; it is expressed as a measure of similarity between the node response to an unknown pattern U and the node responses to the training patterns.

The retrieval equations of the GNU, which give the evolution in time of the overlaps between an unknown pattern U and the stored patterns, are defined. Then, the retrieval equations of the GNU network are derived in 2 particular cases.

7.2. Relationships between pattern overlaps

7.2.1. Principle of inclusion and exclusion

The Principle of Inclusion and Exclusion (or PIE) [PolTW83] is useful to derive various important expressions between pattern overlaps.

A set of N objects is considered, which have various properties $i_1, i_2, \dots, i_k$. Each of the objects may have any or none of the properties. Let N_{i_1} be the number of objects that have property i_1 . Some of these objects may have other properties in addition to property i_1 . Similarly, let N_{i_2} be the number of objects that have property i_2 , and so on. Let N_{i_1, i_2} be the number of objects that have *both* property i_1 and property i_2 . N_{i_1, i_2} the number that have properties i_1 and i_2 , etc. $N_{i_1, i_2, \dots, i_k}$ is the number of objects that have *all* the properties. The PIE consists of a general formula for computing N_0 , the number of objects that have *none* of the properties [PolTW83]:

$$\begin{aligned}
N_0 = & N - N_{i_1} - N_{i_2} - N_{i_3} - \dots - N_{i_k} \\
& + N_{i_1 i_2} + N_{i_1 i_3} + N_{i_2 i_3} + \dots + N_{i_1 i_2 i_3} \\
& - N_{i_1 i_2 i_3} - N_{i_1 i_2 i_4} - \dots \\
& \vdots \\
& + (-1)^K * N_{i_1 i_2 i_3 \dots i_K}
\end{aligned} \tag{7.1}$$

Let $m^{i_1 i_2 \dots i_k}$ be the number of pixels of the GNU feed-back input terminal which have the same value in patterns $T_{j_1}, T_{j_2}, \dots, T_{j_k}$. Let i_1 be the property that a pixel has the same value in $T_{j_1}, T_{j_2}, \dots, T_{j_k}$, and T_{i_1} . Similarly, let i_2 be the property that a pixel has the same value in $T_{j_1}, T_{j_2}, \dots, T_{j_k}$, and T_{i_2} . And so on for properties $i_3, i_4, \dots, i_k$.

The pixels which have the same value in patterns $T_{j_1}, T_{j_2}, \dots, T_{j_k}$, may have any or none of the properties $i_1 i_2 \dots i_k$. Therefore, using (7.1), the following expression can be written:

$$\begin{aligned}
m_0^{j_1 j_2 \dots j_k} = & m^{j_1 j_2 \dots j_k} - m_{i_1}^{j_1 j_2 \dots j_k} - m_{i_2}^{j_1 j_2 \dots j_k} - \dots - m_{i_k}^{j_1 j_2 \dots j_k} \\
& + m_{i_1 i_2}^{j_1 j_2 \dots j_k} + m_{i_1 i_3}^{j_1 j_2 \dots j_k} + \dots + m_{i_1 i_2 i_3}^{j_1 j_2 \dots j_k} \\
& - m_{i_1 i_2 i_3}^{j_1 j_2 \dots j_k} - m_{i_1 i_2 i_4}^{j_1 j_2 \dots j_k} - \dots \\
& \vdots \\
& + (-1)^K * m_{i_1 i_2 i_3 \dots i_K}^{j_1 j_2 \dots j_k}
\end{aligned} \tag{7.2}$$

with $m_0^{j_1 j_2 \dots j_k}$ the number of pixels which have the same value in patterns $T_{j_1}, T_{j_2}, \dots, T_{j_k}$ and which have a different value in $T_{i_1}, T_{i_2}, \dots, T_{i_k}$. Dividing, in (7.2), each quantity by m , the total number of pixels, and using the notations

$$\frac{1}{m} * m_{i_1 i_2 \dots i_k}^{j_1 j_2 \dots j_k} = \omega_{j_1 j_2 \dots j_k, i_1 i_2 \dots i_k}$$

and

$$\frac{1}{m} * m_0^{j_1 j_2 \dots j_k} = \omega_{j_1 j_2 \dots j_k, \overline{i_1 i_2 \dots i_k}}$$

the following expression between overlaps can be written:

Theorem 7.1: Considering a set of patterns $\{T_{j_1}, T_{j_2}, \dots, T_{j_k}, T_{i_1}, T_{i_2}, \dots, T_{i_k}\}$, the proportion $\omega_{j_1 j_2 \dots j_k, \overline{i_1 i_2 \dots i_k}}$ of the pattern pixels which have the same value in $\{T_{j_1}, T_{j_2}, \dots, T_{j_k}\}$ and a different value in $\{T_{i_1}, T_{i_2}, \dots, T_{i_k}\}$ can be expressed by the following linear relation between pattern overlaps:

$$\begin{aligned}
\omega_{j_1, j_2, \dots, j_k, \overline{i_1 i_2 i_3 \dots i_k}} &= \omega_{j_1, j_2, \dots, j_k} - \omega_{j_1, j_2, \dots, j_k, i_1} - \omega_{j_1, j_2, \dots, j_k, i_2} - \dots - \omega_{j_1, j_2, \dots, j_k, i_k} \\
&\quad + \omega_{j_1, j_2, \dots, j_k, i_1, i_2} + \omega_{j_1, j_2, \dots, j_k, i_1, i_3} + \dots + \omega_{j_1, j_2, \dots, j_k, i_1, i_k} \\
&\quad + \omega_{j_1, j_2, \dots, j_k, i_2, i_3} + \omega_{j_1, j_2, \dots, j_k, i_2, i_4} + \dots \\
&\quad + (-1)^K \cdot \omega_{j_1, j_2, \dots, j_k, i_1, i_2, i_3, \dots, i_k}
\end{aligned} \tag{7.3}$$

Proof: The proof of (7.3) follows immediately from (7.2).

For example,

$$\omega_{1, \overline{23}} = \omega_1 - \omega_{12} - \omega_{13} + \omega_{123} = 1 - \omega_{12} - \omega_{13} + \omega_{123}$$

7.2.2. Complementary overlaps

Due the binary values taken by the pattern pixels, it follows that:

Theorem 7.2: Considering a set of patterns $\{T_{j_1}, T_{j_2}, \dots, T_{j_k}, T_{i_1}, T_{i_2}, \dots, T_{i_k}\}$, the proportion $\omega_{j_1, j_2, \dots, j_k, \overline{i_1 i_2 i_3 \dots i_k}}$ of the pattern pixels which have the same value in $\{T_{j_1}, T_{j_2}, \dots, T_{j_k}\}$ and a different value in $\{T_{i_1}, T_{i_2}, \dots, T_{i_k}\}$, is equal to the proportion $\omega_{i_1 i_2 i_3 \dots i_k, \overline{j_1 j_2 \dots j_k}}$ of the pattern pixels which have the same value in $\{T_{i_1}, T_{i_2}, \dots, T_{i_k}\}$ and a different value in $\{T_{j_1}, T_{j_2}, \dots, T_{j_k}\}$:

$$\omega_{j_1, j_2, \dots, j_k, \overline{i_1 i_2 i_3 \dots i_k}} = \omega_{i_1 i_2 i_3 \dots i_k, \overline{j_1 j_2 \dots j_k}} \tag{7.4}$$

Proof: This is a direct consequence of the fact that a pixel can only take the value 0 or the value 1. Therefore a pixel accounted for in $m_{j_1 j_2 \dots j_k}^{i_1 i_2 \dots i_k}$ will also be accounted for in $m_{i_1 i_2 \dots i_k}^{j_1 j_2 \dots j_k}$. Equation (7.4) follows.

7.2.3. Useful corollaries

Corollary 7.1:

$$\omega_{ijk} = \frac{1}{2}(-1 + \omega_{ij} + \omega_{ik} + \omega_{jk}) \tag{7.5}$$

Proof: From (7.4), it follows that

$$\omega_{i\bar{j}\bar{k}} = \omega_{jk\bar{i}}. \quad (7.6)$$

By developing each side of (7.6), using (7.3), it holds that

$$\omega_i - \omega_{ij} - \omega_{ik} + \omega_{ijk} = \omega_{jk} - \omega_{ijk}.$$

Corollary 7.2:

$$\omega_{i\bar{j}\bar{k}} + \omega_{ijk} + \omega_{ik\bar{j}} + \omega_{jk\bar{i}} = 1 \quad (7.7)$$

Proof: By developing each term on the left-hand side of (7.7), using (7.3), the identity (7.7) follows immediately.

7.2.4. Higher order overlaps

Corollary 7.1 expresses the overlap between 3 patterns as a linear function of the overlaps between any 2 of these patterns. The general case of the overlap $\omega_{i_1 i_2 \dots i_\kappa}$ between patterns $T_{i_1}, T_{i_2}, \dots, T_{i_\kappa}$ is considered. The variable κ is called the *order* of the overlap $\omega_{i_1 i_2 \dots i_\kappa}$.

Theorem 7.3: *The overlap between a odd number of patterns can be expressed as a linear function of lower order overlaps. The overlap between a even number of patterns cannot be expressed as a function of lower order overlaps.*

Proof: Using Theorem 7.2, a number of relations can be written, which take the form:

$$\omega_{i_1 i_2 \dots i_\kappa \overline{i_{\lambda+1} i_{\lambda+2} \dots i_\kappa}} = \omega_{i_1 i_2 \dots i_\lambda \overline{i_{\lambda+1} i_{\lambda+2} \dots i_\kappa}}. \quad (7.8)$$

with $\lambda = 2, 3, \dots, \kappa - 1$.

Theorem 7.1 is applied to each side of (7.8), which yields:

$$\begin{aligned} \omega_{i_1 i_2 \dots i_\kappa \overline{i_{\lambda+1} i_{\lambda+2} \dots i_\kappa}} &= \omega_{i_1 i_2 \dots i_\lambda} - \omega_{i_1 i_2 \dots i_\lambda i_{\lambda+1}} - \omega_{i_1 i_2 \dots i_\lambda i_{\lambda+2}} - \dots - \omega_{i_1 i_2 \dots i_\lambda i_\kappa} \\ &\quad + \omega_{i_1 i_2 \dots i_\lambda i_{\lambda+1} i_{\lambda+2}} + \omega_{i_1 i_2 \dots i_\lambda i_{\lambda+1} i_{\lambda+3}} + \dots + \omega_{i_1 i_2 \dots i_\lambda i_{\lambda+1} i_\kappa} \\ &\quad - \omega_{i_1 i_2 \dots i_\lambda i_{\lambda+1} i_{\lambda+2} i_{\lambda+3}} - \omega_{i_1 i_2 \dots i_\lambda i_{\lambda+1} i_{\lambda+2} i_{\lambda+4}} - \dots \\ &\quad \vdots \\ &\quad + (-1)^{\kappa-\lambda} \cdot \omega_{i_1 i_2 \dots i_\kappa} \end{aligned}$$

and

$$\begin{aligned} \omega_{i_1, \dots, i_{\kappa-2}, i_{\kappa-1}, i_{\kappa}} \overline{i_1 i_2 \dots i_{\kappa}} &= \omega_{i_1, \dots, i_{\kappa-2}, i_{\kappa-1}, i_{\kappa}} - \omega_{i_1, i_2, \dots, i_{\kappa-2}, i_{\kappa-1}, i_{\kappa}} - \omega_{i_1, i_2, \dots, i_{\kappa-2}, i_{\kappa-1}, i_{\kappa}} - \dots - \omega_{i_1, i_2, \dots, i_{\kappa-2}, i_{\kappa-1}, i_{\kappa}} \\ &\quad + \omega_{i_1, i_2, i_3, \dots, i_{\kappa-2}, i_{\kappa-1}, i_{\kappa}} + \omega_{i_1, i_2, i_3, \dots, i_{\kappa-2}, i_{\kappa-1}, i_{\kappa}} + \dots + \omega_{i_1, i_2, i_3, \dots, i_{\kappa-2}, i_{\kappa-1}, i_{\kappa}} \\ &\quad - \omega_{i_1, i_2, i_3, i_4, \dots, i_{\kappa-2}, i_{\kappa-1}, i_{\kappa}} - \omega_{i_1, i_2, i_3, i_4, \dots, i_{\kappa-2}, i_{\kappa-1}, i_{\kappa}} - \dots \\ &\quad \vdots \\ &\quad + (-1)^{\lambda} \cdot \omega_{i_1, i_2, \dots, i_{\kappa}}. \end{aligned}$$

Equation (7.8) becomes:

$$\begin{aligned} \omega_{i_1, i_2, \dots, i_{\kappa}} &= [(-1)^{\kappa-\lambda} - (-1)^{\lambda}]^{-1} \cdot \left\{ \left(\omega_{i_1, i_2, \dots, i_{\kappa-2}, i_{\kappa-1}, i_{\kappa}} - \omega_{i_1, i_2, \dots, i_{\kappa-2}, i_{\kappa-1}, i_{\kappa}} - \omega_{i_1, i_2, \dots, i_{\kappa-2}, i_{\kappa-1}, i_{\kappa}} - \dots - \omega_{i_1, i_2, \dots, i_{\kappa-2}, i_{\kappa-1}, i_{\kappa}} \right. \right. \\ &\quad \left. \left. + \omega_{i_1, i_2, i_3, \dots, i_{\kappa-2}, i_{\kappa-1}, i_{\kappa}} + \omega_{i_1, i_2, i_3, \dots, i_{\kappa-2}, i_{\kappa-1}, i_{\kappa}} + \dots + \omega_{i_1, i_2, i_3, \dots, i_{\kappa-2}, i_{\kappa-1}, i_{\kappa}} \right. \right. \\ &\quad \left. \left. - \omega_{i_1, i_2, i_3, i_4, \dots, i_{\kappa-2}, i_{\kappa-1}, i_{\kappa}} - \omega_{i_1, i_2, i_3, i_4, \dots, i_{\kappa-2}, i_{\kappa-1}, i_{\kappa}} - \dots \right) \right. \\ &\quad \left. - \left(\omega_{i_1, i_2, \dots, i_{\kappa}} - \omega_{i_1, i_2, \dots, i_{\kappa-1}, i_{\kappa}} - \omega_{i_1, i_2, \dots, i_{\kappa-1}, i_{\kappa}} - \dots - \omega_{i_1, i_2, \dots, i_{\kappa-1}, i_{\kappa}} \right. \right. \\ &\quad \left. \left. + \omega_{i_1, i_2, \dots, i_{\kappa-1}, i_{\kappa-1}, i_{\kappa}} + \omega_{i_1, i_2, \dots, i_{\kappa-1}, i_{\kappa-1}, i_{\kappa}} + \dots + \omega_{i_1, i_2, \dots, i_{\kappa-1}, i_{\kappa-1}, i_{\kappa}} \right. \right. \\ &\quad \left. \left. - \omega_{i_1, i_2, \dots, i_{\kappa-1}, i_{\kappa-1}, i_{\kappa-1}, i_{\kappa}} - \omega_{i_1, i_2, \dots, i_{\kappa-1}, i_{\kappa-1}, i_{\kappa-1}, i_{\kappa}} - \dots \right) \right\}. \end{aligned} \quad (7.9)$$

In the right-hand side of (7.9), the first factor can be transformed as

$$[(-1)^{\kappa-\lambda} - (-1)^{\lambda}]^{-1} = (-1)^{-\lambda} \cdot [(-1)^{\kappa-2\lambda} - 1]^{-1},$$

which shows that it is only defined if κ is an odd integer, and its value in that case is:

$$\frac{1}{2} \cdot (-1)^{\lambda+1},$$

Therefore, (7.9) is only defined when the order κ of the overlap $\omega_{i_1, i_2, \dots, i_{\kappa}}$ is odd. This concludes of proof of Theorem 7.3.

7.3. Retrieval Process in the GNU network

7.3.1. Definitions

7.3.1.1. Spreading function

It is assumed, in the following, that a spreading process, with full generalisation, takes place in the nodes of an auto-associative GNU network after training. The spreading function is expressed, for a GNU node, as a function of the similarity between the node response to an unknown pattern U , and the node responses to a number S of training patterns $T_1, T_2, \dots, T_S$. The superscript v is used to indicate that the samples considered refer to the input and output of node v .

When a sample U^v of pattern U is presented to node v of the GNU, the node will output a value corresponding to one or more of the training patterns $T_1, T_2, \dots, T_S$. Using n_{iU}^v to denote the number of inputs to node v that overlap in U^v and T_i^v , the overlap ω_{iU}^v between U^v and T_i^v can be expressed as

$$\omega_{iU}^v = \frac{n_{iU}^v}{F}$$

with F the number of feed-back inputs to the GNU and

$$0 < n_{iU}^v < F.$$

The set of indices

$$I_{\max} = \{i_1, i_2, \dots, i_j, \dots, i_p\}$$

is defined as referring to the pattern samples $T_{i_1}^v, T_{i_2}^v, \dots, T_{i_p}^v$ which have the maximum overlap with U^v , such that

$$\forall i_j \in I_{\max}: n_{i_j U}^v = \max_{k=1}^S n_{kU}^v,$$

Also, the notation $\omega_{i_1, i_2, \dots, i_p}^v$ is used to represent the overlap between pattern samples $T_{i_1}, T_{i_2}, \dots, T_{i_p}$ at the output of node v , with

$$\omega_{i_1, i_2, \dots, i_p}^v = 1$$

if the output of node v maps into an area of overlap between patterns $T_{i_1}, T_{i_2}, \dots, T_{i_h}$ and

$$\omega_{i_1, i_2, \dots, i_h}^v = 0$$

otherwise. Using $y^v(T_j)$ to denote the output value at node v in response to input sample T_j , the output function of the GNU nodes, after spreading has taken place, is then given by:

$$y^v(U) = \begin{cases} y^v(T_{i_1}) & \text{for } \omega_{i_1, i_2, \dots, i_p}^v = 1 \\ \text{1 or 0, chosen randomly} & \text{for } \omega_{i_1, i_2, \dots, i_p}^v = 0 \end{cases} \quad (7.10)$$

with

$$i_1, i_2, \dots, i_p \in I_{\max}$$

7.3.1.2. Retrieval equations

The retrieval equations of the GNU give the evolution in time of the overlaps between an unknown pattern U , presented to the GNU, and the S stored patterns $T_1, T_2, \dots, T_S$. The notation

$$\omega_{i_1, i_2, \dots, i_h} = \omega_{[b(i_1, i_2, \dots, i_h)]}$$

is used, with

$$b(i_1, i_2, \dots, i_h) = \sum_{c=1}^h 2^{i_c-1}.$$

For example,

$$b(3, 4, 6) = 2^2 + 2^3 + 2^5 = 44$$

and therefore

$$\omega_{345} = \omega_{[44]}.$$

In general, S retrieval equations are necessary in order to describe the retrieval process

taking place in the GNU:

$$\omega_{iu}(t+1) = f_i(\omega_{11}, \omega_{12}, \dots, \omega_{12^{x-1}}, \omega_{1u}(t), \omega_{2u}(t), \dots, \omega_{su}(t)), \quad (7.11)$$

with $i = 1, 2, \dots, S$.

For example, in the case of 3 training patterns T_1, T_2, T_3 , (7.11) becomes:

$$\begin{cases} \omega_{1u}(t+1) = f_1(\omega_{12}, \omega_{13}, \omega_{23}, \omega_{1u}(t), \omega_{2u}(t), \omega_{3u}(t)) \\ \omega_{2u}(t+1) = f_2(\omega_{12}, \omega_{13}, \omega_{23}, \omega_{1u}(t), \omega_{2u}(t), \omega_{3u}(t)) \\ \omega_{3u}(t+1) = f_3(\omega_{12}, \omega_{13}, \omega_{23}, \omega_{1u}(t), \omega_{2u}(t), \omega_{3u}(t)) \end{cases} \quad (7.12)$$

This case will be investigated in detail in section 7.3.3. Often, in the subsequent sections, use is made of the notation

$$\omega_{iu} = x_i, \quad (7.13)$$

7.3.2. Retrieval of two opposite patterns

A GNU trained on two opposite patterns T_1 and T_2 is considered. It is assumed that a spreading process, with full generalisation, takes place in the network nodes after training. At time t , the GNU is in a state that has an overlap $x(t)$ with pattern T_1 , and therefore an overlap $1 - x(t)$ with pattern T_2 . This system has been studied in [Ale90c]. Equations (7.12) reduce here to a single equation. Using (7.13) and the fact that the 2 training patterns are opposite, that is,

$$x_1 = 1 - x_2 = x,$$

the retrieval equation of the system can be stated as:

$$x(t+1) = f(x(t), F)$$

$$= \begin{cases} \sum_{j=\frac{F}{2}}^F \binom{F}{j} x^j(t) (1-x(t))^{F-j} & \text{for } F \text{ odd} \\ \sum_{j=\frac{F}{2}-1}^F \binom{F}{j} x^j(t) (1-x(t))^{F-j} + \frac{1}{2} \binom{F}{\frac{F}{2}} x^{\frac{F}{2}}(t) (1-x(t))^{\frac{F}{2}} & \text{for } F \text{ even} \end{cases} \quad (7.14)$$

where $\lceil z \rceil$ is the ceiling of z . In [Ale90c], it is shown that the system enters the trained state which has the greater overlap with the initial state of the GNU. The larger the number of inputs F , the more rapidly the system will converge.

As a contribution to this thesis, the following result has been derived:

Theorem 7.4: *The retrieval equation of an auto-associative GNU storing 2 opposite patterns is the same for a GNU with a number of feedback inputs per node $F = 2n$ and for a GNU with $F = 2n - 1$, with n being an integer greater than 0:*

$$f(x(t), 2n) = f(x(t), 2n - 1). \quad (7.15)$$

Proof: A proof of (7.15) is given in Appendix E.

Using (7.15), the expressions (7.14) reduce to a single equation, which applies to both odd and even values of F :

$$\begin{aligned} x(t+1) &= f(x(t), F) \\ &= f(x(t), 2 \cdot \left\lceil \frac{F}{2} \right\rceil - 1) = \sum_{j=\lceil \frac{F}{2} \rceil}^{2 \cdot \left\lceil \frac{F}{2} \rceil - 1} \left(2 \cdot \left\lceil \frac{F}{2} \right\rceil - 1 \right) \cdot x^j(t) \cdot (1 - x(t))^{2 \cdot \left\lceil \frac{F}{2} \right\rceil - j - 1}. \end{aligned} \quad (7.16)$$

7.3.3. General retrieval of three patterns

In this sub-section, the equations governing the retrieval process of the system are established in the case of three stored patterns. Again, after training, a spreading phase, with full generalisation, takes place in the nodes. An unknown pattern U is presented to the network. Let the variables $x_1(t), x_2(t), x_3(t)$ represent the overlaps, at time t , between pattern U and patterns T_1, T_2 , and T_3 , respectively. The evolution through time of $x_1(t), x_2(t)$, and $x_3(t)$ is monitored.

Considering a particular F -input node, the number of inputs that have the same values for U and T_i is denoted n_j . In Figure 7.1, the patterns T_1, T_2, T_3 are represented as sets in a Venn diagram. In each region of the Venn diagram, the relative proportions of pixels that belong to that region, are indicated. Equivalently, these proportions can be seen as the probabilities that an input or output to a node takes its value from that region. For example, if corresponding pixels in patterns T_1, T_2 , and T_3 , are compared, α is the proportion of pixels in pattern T_1 that have opposite values in patterns T_2 , and T_3 , as represented in Figure 7.1.

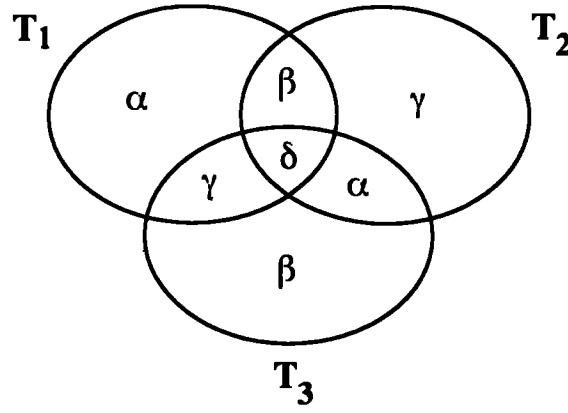


Fig. 7.1: Patterns are represented in a Venn diagram in which the proportional areas associated with each region are indicated.

The overlaps ω_{ij} between training patterns can be expressed in terms of the probabilities $\alpha, \beta, \gamma, \delta$:

$$\begin{cases} \omega_{12} = \beta + \delta \\ \omega_{13} = \gamma + \delta \\ \omega_{23} = \alpha + \delta \end{cases} \quad (7.17)$$

with

$$\alpha + \beta + \gamma + \delta = 1. \quad (7.18)$$

The quantities α, β, γ and δ are the overlaps $\omega_{1,23}, \omega_{12,3}, \omega_{13,2}$ and ω_{123} , respectively, and (7.18) is identical to (7.7). It holds that:

$$\begin{cases} \alpha = \frac{1}{2}(1 - \omega_{12} - \omega_{13} + \omega_{23}) \\ \beta = \frac{1}{2}(1 + \omega_{12} - \omega_{13} - \omega_{23}) \\ \gamma = \frac{1}{2}(1 - \omega_{12} + \omega_{13} - \omega_{23}) \\ \delta = \frac{1}{2}(-1 + \omega_{12} + \omega_{13} + \omega_{23}). \end{cases} \quad (7.19)$$

In the case of S training patterns, (7.19) becomes a set of 2^{S-1} equations.

In Figure 7.2, the areas of intersection of patterns T_1, T_2, T_3 with pattern U are shown. This defines the proportion variables ξ, η, θ , and λ . For example, a proportion ξ of the pixels in region α will have the same value for patterns T_1 and U , and so a

proportion $1 - \xi$ will have the same value for patterns T_2 , T_1 , and U . This holds true for the other regions. In fact, ξ , η , θ , and λ can be expressed as:

$$\begin{cases} \omega_{1U \cdot 23} = \xi \cdot \omega_{1 \cdot 23} \\ \omega_{2U \cdot 13} = \eta \cdot \omega_{2 \cdot 13} \\ \omega_{3U \cdot 12} = \theta \cdot \omega_{3 \cdot 12} \\ \omega_{123U} = \lambda \cdot \omega_{123} \end{cases}$$

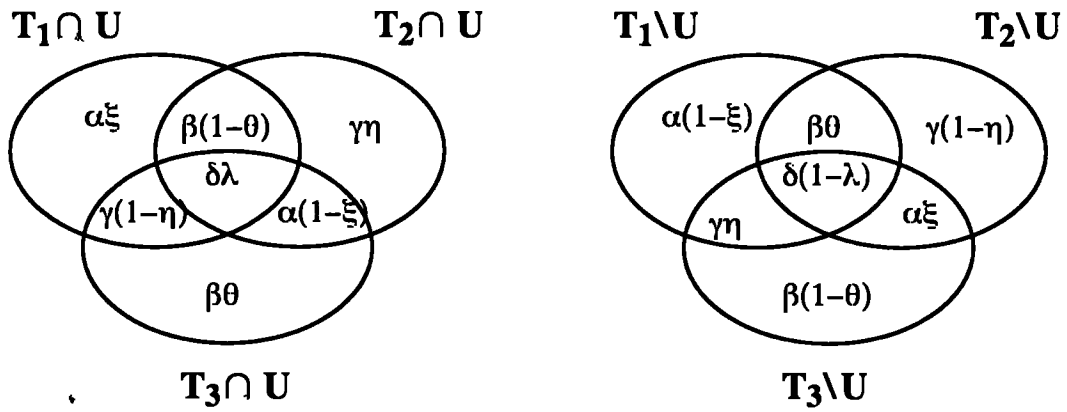


Fig. 7.2: Areas of intersection and difference between patterns T_1 , T_2 , T_3 and U .

Dropping, for the time being, the dependency on t_i it is straightforward to write the relations

$$\begin{cases} x_1 = \xi\alpha + (1-\theta)\beta + (1-\eta)\gamma + \lambda\delta \\ x_2 = (1-\xi)\alpha + (1-\theta)\beta + \eta\gamma + \lambda\delta \\ x_3 = (1-\xi)\alpha + \theta\beta + (1-\eta)\gamma + \lambda\delta \end{cases} \quad (7.20)$$

Expression (7.20) is a linear system of 3 equations with 4 unknowns variables ξ , η , θ , and λ . If λ is chosen as an arbitrary parameter, the solution of the system is:

$$\begin{cases} \xi = \frac{1}{2\alpha}(2\alpha + \beta + \gamma + 2\lambda\delta - x_2 - x_3) \\ \eta = \frac{1}{2\gamma}(\alpha + \beta + 2\gamma + 2\lambda\delta - x_1 - x_3) \\ \theta = \frac{1}{2\beta}(\alpha + 2\beta + \gamma + 2\lambda\delta - x_1 - x_2) \end{cases} \quad (7.21)$$

A spreading function, as defined by (7.10) is used here. There are 7 different cases to be

considered, corresponding to the number of ways of forming the set $I_{\max}$ (defined in Section 7.3.1.1) with 3 indices. In the case of S training patterns, $2^S - 1$ cases would need to be considered. The probabilities of occurrence $p_1, p_2, \dots, p_7$ of the 7 present cases can be expressed as:

$$\begin{cases} p_1 = \Pr[(n_{1U}^v > n_{2U}^v) \wedge (n_{1U}^v > n_{3U}^v)] \\ p_2 = \Pr[(n_{2U}^v > n_{1U}^v) \wedge (n_{2U}^v > n_{3U}^v)] \\ p_3 = \Pr[(n_{3U}^v > n_{1U}^v) \wedge (n_{3U}^v > n_{2U}^v)] \\ p_4 = \Pr[(n_{1U}^v = n_{2U}^v) \wedge (n_{1U}^v > n_{3U}^v)] \\ p_5 = \Pr[(n_{1U}^v = n_{3U}^v) \wedge (n_{1U}^v > n_{2U}^v)] \\ p_6 = \Pr[(n_{2U}^v = n_{3U}^v) \wedge (n_{2U}^v > n_{1U}^v)] \\ p_7 = \Pr[(n_{1U}^v = n_{2U}^v) \wedge (n_{1U}^v = n_{3U}^v)] \end{cases} \quad (7.22)$$

with $\wedge$ representing the AND connective and

$$\sum_i p_i = 1. \quad (7.23)$$

The exact formulae yielding the probabilities p_i are sums over a multinomial distribution [GriS82], whose parameters are the variables associated with the different regions represented in Figure 7.2 (b). That is, for example,

$$\begin{aligned} p_1 = \sum_{\substack{0 \leq q_i \leq F \\ q_1 + q_6 \geq q_2 + q_5 \\ q_1 + q_4 \geq q_2 + q_3}} \frac{F!}{q_1! q_2! \dots q_8!} [\alpha \xi]^{q_1} \cdot [\alpha(1 - \xi)]^{q_2} \cdot [\beta \theta]^{q_3} \cdot [\beta(1 - \theta)]^{q_4} \cdot \\ [\gamma \eta]^{q_5} \cdot [\gamma(1 - \eta)]^{q_6} \cdot [\delta \lambda]^{q_7} \cdot [\delta(1 - \lambda)]^{q_8}, \end{aligned} \quad (7.24)$$

with

$$\sum_{i=1}^8 q_i = F$$

and assuming that none of the variables under brackets is equal to 0 (in which case (7.24) would have to be rewritten without the corresponding q_i). In (7.24), the second and third conditions under which the sum is taken, correspond to the condition under which p_1 is expressed in (7.22).

The variable x_1 can be expressed as a function of the probabilities p_i and variables $\alpha, \beta, \gamma, \delta$. This yields:

$$\begin{aligned}
 x_1 = & p_1 \cdot (\alpha + \beta + \gamma + \delta) + p_2 \cdot (\beta + \delta) + p_3 \cdot (\gamma + \delta) + p_4 \cdot (\beta + \delta + \frac{1}{2}(\alpha + \gamma)) \\
 & + p_5 \cdot (\gamma + \delta + \frac{1}{2}(\alpha + \beta)) + p_6 \cdot (\delta + \frac{1}{2}(\beta + \gamma)) + p_7 \cdot (\delta + \frac{1}{2}(\alpha + \beta + \gamma))
 \end{aligned} \quad (7.25)$$

Similar expressions to (7.25) are obtained for x_2 and x_3 . Then, using, for example, the first equation in (7.20) in conjunction with (7.25), the values of $\xi, \eta, \theta, \lambda$ can be identified as:

$$\left\{ \begin{array}{l}
 \alpha: \quad \xi = p_1 + \frac{1}{2}p_4 + \frac{1}{2}p_5 + \frac{1}{2}p_7 \\
 \beta: \quad 1 - \theta = p_1 + p_2 + p_4 + \frac{1}{2}p_5 + \frac{1}{2}p_6 + \frac{1}{2}p_7 \\
 \gamma: \quad 1 - \eta = p_1 + p_3 + p_5 + \frac{1}{2}p_4 + \frac{1}{2}p_6 + \frac{1}{2}p_7 \\
 \delta: \quad \lambda = p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7
 \end{array} \right.$$

The evolution of $\xi, \eta, \theta, \lambda$ in time is therefore given by:

$$\left\{ \begin{array}{l}
 \xi(t+1) = p_1(t) + \frac{1}{2}(p_4(t) + p_5(t) + p_7(t)) \\
 \eta(t+1) = p_2(t) + \frac{1}{2}(p_4(t) + p_6(t) + p_7(t)) \\
 \theta(t+1) = p_3(t) + \frac{1}{2}(p_5(t) + p_6(t) + p_7(t)) \\
 \lambda(t+1) = 1,
 \end{array} \right. \quad (7.26)$$

for $t = 0, 1, 2, \dots$. It is worth noting the last equation in expression (7.26), which shows that λ takes the value 1 from time $t = 1$ onwards, independently of the value of λ at time $t = 0$. Indeed, due to the spreading function used, all pixels which have the same value in T_1, T_2, T_3 and which, initially, had the opposite value in U (these pixels were accounted for in the overlap $(1 - \lambda)\delta$) will, after one time step, have the same value in all 4 patterns T_1, T_2, T_3 and U .

The expressions for $\xi, \eta, \theta, \lambda$, given by (7.26), when substituted in (7.20), yield the retrieval equations of the system. The inter-dependence among the sets of variables, determining the system evolution in time, can be best understood from the simple flow

chart shown in Figure 7.3.

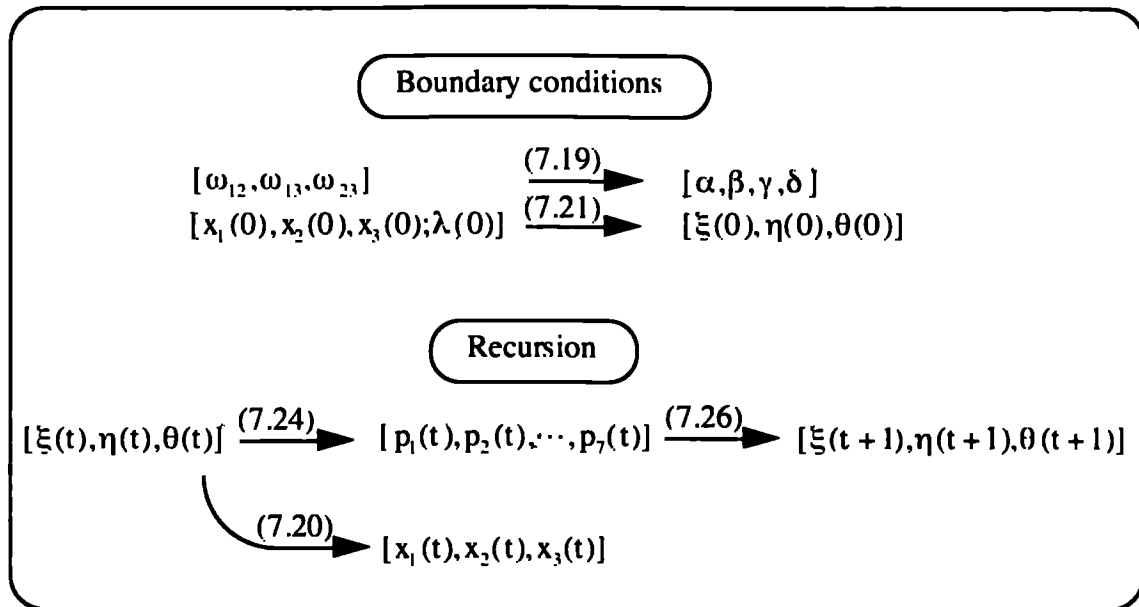


Fig. 7.3: Flow chart showing the interdependence among the sets of variables determining the system evolution in time.

The retrieval equations are a useful tool for studying the dynamics of the system, the stable states, and the extent of the basins of attraction.

7.3.4. Example

The quantities intervening in the equations of the previous sub-section are now calculated, using an example. The example patterns chosen are represented in Figure 7.4.

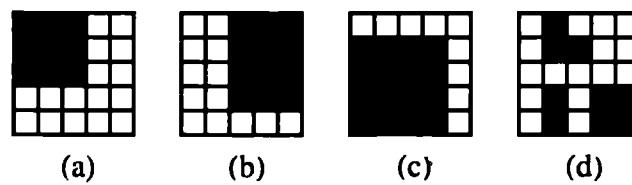


Fig. 7.4: Example patterns (a) T_1 , (b) T_2 , (c) T_3 and (d) U .

The following quantities can then be calculated:

$$\omega_{12} = \frac{10}{25}; \omega_{13} = \frac{12}{25}; \omega_{23} = \frac{9}{25}.$$

$$x_1 = w_{1U} = \frac{13}{25}; \quad x_2 = w_{2U} = \frac{10}{25}; \quad x_3 = w_{3U} = \frac{12}{25};$$

$$\delta = w_{123} = \frac{1}{2}(-1 + w_{12} + w_{13} + w_{23}) = \frac{3}{25};$$

$$\alpha = w_{1\bar{2}\bar{3}} = 1 - w_{12} - w_{13} + w_{123} = \frac{6}{25};$$

$$\beta = w_{3\bar{1}\bar{2}} = 1 - w_{13} - w_{23} + w_{123} = \frac{7}{25};$$

$$\gamma = w_{2\bar{1}\bar{3}} = 1 - w_{13} - w_{23} + w_{123} = \frac{9}{25};$$

$$\lambda\delta = w_{123U} = \frac{1}{25} \longrightarrow \lambda = \frac{\lambda\delta}{\delta} = \frac{w_{123U}}{w_{123}} = \frac{1}{3};$$

$$\alpha\xi = w_{1U\bar{2}\bar{3}} = \frac{4}{25} \longrightarrow \xi = \frac{\alpha\xi}{\alpha} = \frac{2}{3};$$

$$\beta\theta = w_{3U\bar{1}\bar{2}} = \frac{4}{25} \longrightarrow \theta = \frac{\beta\theta}{\beta} = \frac{4}{7};$$

$$\gamma\eta = w_{2U\bar{1}\bar{3}} = \frac{4}{25} \longrightarrow \eta = \frac{\gamma\eta}{\gamma} = \frac{4}{9}.$$

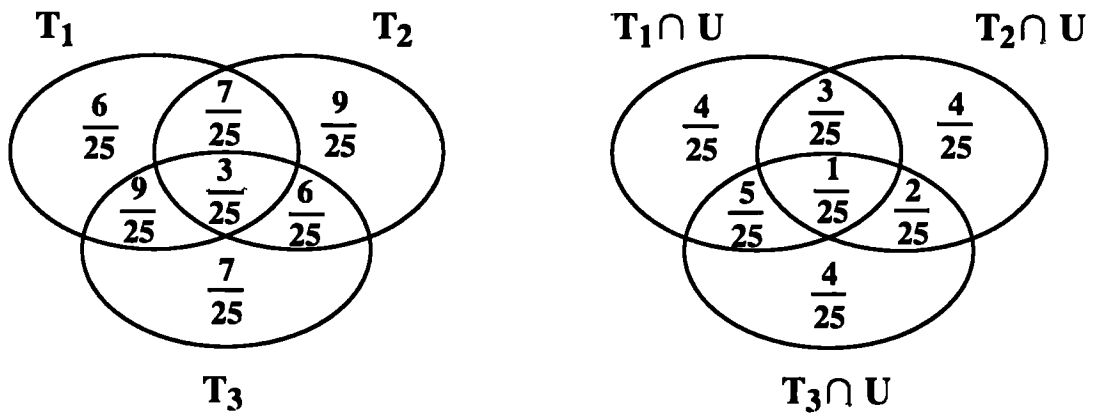


Fig. 7.5: Representation, in Venn diagrams, of the areas of intersection between the example patterns T_1 , T_2 , T_3 and U , represented in Figure 7.6.

7.4. Conclusions

The retrieval process in the GNU network has been investigated. First, the relationships between pattern overlaps were derived and several useful corollaries were formulated, using the principle of inclusion and exclusion. One of the results was that the overlap between a odd number of patterns could be expressed as a linear function of lower order overlaps, whereas no such expression existed in the case of the overlap between an even number of patterns.

The retrieval equations of the GNU network were first defined and then established in the cases of two opposite stored patterns and three arbitrary stored patterns. These equations, even for 3 arbitrary stored patterns, form complicated expressions between pattern overlaps. They govern the dynamics of the system.

CHAPTER VIII. DYNAMICALLY GENERALISING WEIGHTLESS NEURAL MEMORIES

8.1. Introduction

The concept of the *Dynamically Generalising* Random Access Memory (DGRAM) is introduced. The DGRAM is derived from the GRAM [Ale90a], in which the *training* and *spreading* operations are replaced by a single *learning* phase. The DGRAM is able to store and spread patterns, through a dynamical process involving interactions between each memory location, its immediate neighbours and external signals.

The DGRAM exhibits very desirable properties, compared with those of the GRAM. First, after the initial trained patterns have spread throughout the memory space, additional patterns can still be stored in the DGRAM. Secondly, it is possible to distinguish between trained and spread patterns. And finally, a trained pattern and its associated spread patterns can be removed without affecting the rest of the stored patterns.

The following sections are organised as follows. The DGRAM model is first derived from the GRAM model. The state transitions occurring in the DGRAM are then analysed and the related logical equations are derived. This is followed by two examples and a discussion.

8.2. From GRAM to DGRAM

The GRAM was reviewed in detail in Section 2.3.7. The training and operating phases are the realisations of the IAN's first property, described in Section 2.3.7.1 During the spreading phase, memory locations not addressed during the training phase, are affected by the use of a spreading algorithm, according to the IAN's properties 2 and 3. Figure 8.1 shows the different phases of processing in a GRAM.

The spreading algorithm can take several forms. In Section 8.3.4, a general form is given. The spreading algorithm in the GRAM resembles the diffusion algorithm described in Section 2.3.7.3. The major difference lies in the absence of majority rule for the GRAM. In case of conflict, either during training or spreading, the content of the corresponding memory location is left at, or reset to the u value.

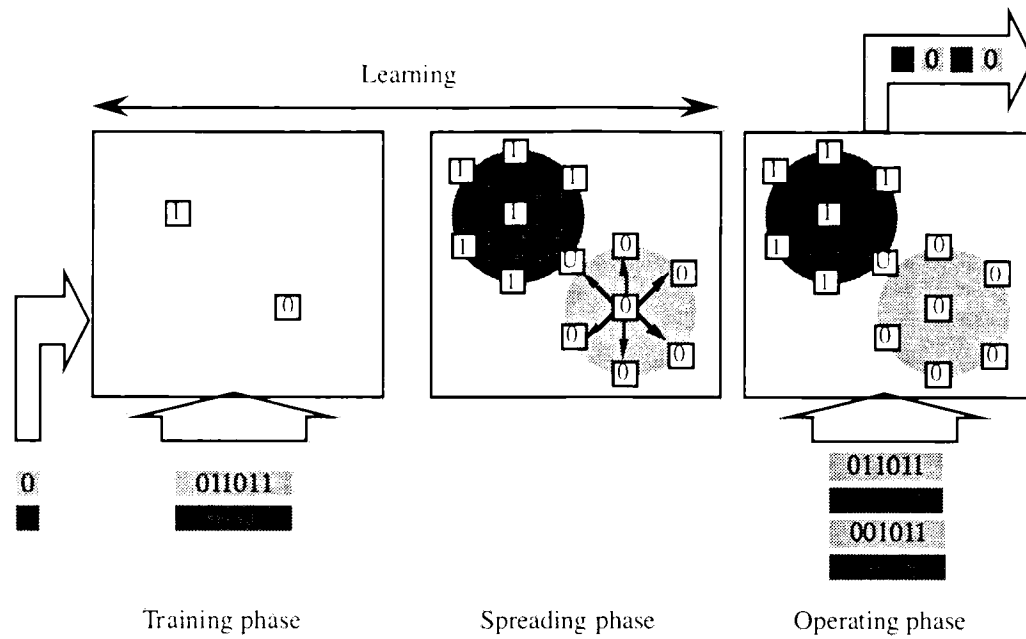


Fig. 8.1: The different phases of processing in a GRAM.

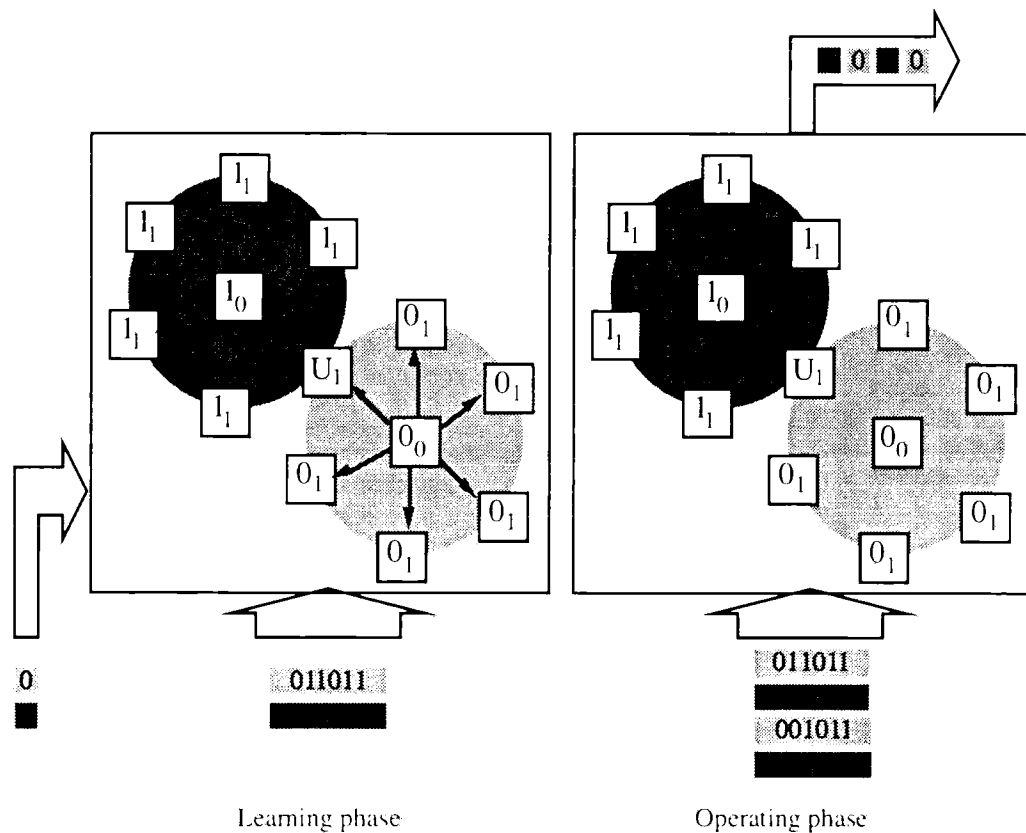


Fig. 8.2: The different phases of processing in a DGRAM.

It is interesting to consider, on one hand, some positive aspects of the GRAM. The spreading can be done "off-line", that is, between the time that training information has been captured and the time that the nodes have to use what was learnt [Ale90a]. Moreover, the GRAM can also be implemented as a *virtual memory* [Mrs93], in which only the patterns from the training set are stored. This is similar to Kanerva's Best Match Machine [Kan88] (see Section 2.3.7.3).

On the other hand, the GRAM does not have some properties that might be desirable: all the training patterns have to be stored first, prior to any spreading. Therefore no new pattern can be stored after spreading; a learnt pattern cannot be distinguished from a spread pattern; it is not possible to remove a learnt pattern and its associated spread patterns.

The following section introduces a weightless node called Dynamically Generalising Random Access Memory (DGRAM) which operates in two phases. During the first phase, patterns from a training set are trained and spread throughout the node. The second phase is the use phase. The DGRAM is able to store patterns and spread them through a dynamical process involving interactions between each memory location and its neighbours, and external signals. Figure 8.2. shows the different phases of processing in a DGRAM.

8.3. The dynamically generalising random access memory

8.3.1. Notations

In the following of this Chapter, the states which a memory location can be in, are denoted by variables such as S , X_i , 1 , 0 or R_i . The font *bold* is used to show that the symbols represent neither scalar nor Boolean values, but, rather, are state identifiers, whose numerical coding is not specified and is of no relevance here.

Additionally, several state transition equations are derived, which involve both state variables X_i and Boolean values z_i . For instance, the equation

$$S' = z_i \cdot S + \bar{z}_i \cdot X_i$$

means that a memory location will, at the next time step, stay in its present state S if the Boolean variable z_i has the value 1 or will transit to state X_i if z_i has the value 0.

8.3.2. State transitions in a RAM

The trivial case of the RAM node is first considered. The contents of a memory location is seen as one of the *internal states* which the location can be in. A memory location can be in one of two states, 1 or 0 . When a RAM learns a new pattern, during a *write* operation, the state transition of a memory location can be expressed as:

$$S' = S \cdot (\bar{w} + \bar{a} \cdot w) + a \cdot w \cdot (x \cdot 1 + \bar{x} \cdot 0), \quad (8.1)$$

in which S and S' are the states of a memory location before and after the write operation, respectively; the Boolean variable w takes the value 1 during a write operation and 0 otherwise; x represents the Boolean data-in value and a is a Boolean variable whose value is 1 if the memory location is addressed and 0 otherwise. It is useful to note the following relation:

$$S = (S = 0) \cdot 0 + (S = 1) \cdot 1 = s_0 \cdot 0 + s_1 \cdot 1$$

or, in a more general form:

$$S = \sum_i s_{X_i} \cdot X_i, \quad (8.2)$$

with

$$s_{X_i} \equiv (S = X_i) \quad (8.3)$$

and in which the X_i are the different states which a memory location can be in. Using the notation

$$S_{X_i} = s_{X_i} \cdot X_i, \quad (8.4)$$

Equation (8.2) can be rewritten as

$$S = \sum_i S_{X_i}. \quad (8.5)$$

Figure 8.3. shows the state transition diagram of a memory location in a RAM, during a write operation. Transitions which do not produce a change of state are not represented. If the state of the addressed location ($a = 1$) is 1 and the data-in value is 0, the state of this location becomes 0. Alternatively, if the state of the addressed location is 0 and the data-in value is 1, the state of this location becomes 1. Otherwise the addressed location remains in the same state. The new state of the addressed location depends solely on the value on the data-in line. Memory locations not addressed are not affected by the write operation. This will no longer be the case for the DGRAM in which the state of a memory location is dynamically updated through interactions with neighbouring locations.

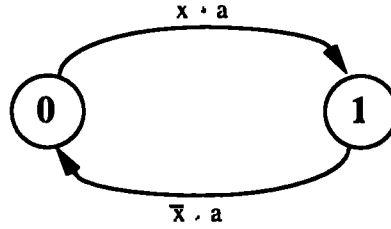


Fig. 8.3: State transition diagram of a memory location in a RAM, during a write operation ($w = 1$).

During a *read* operation, the RAM outputs 0 if a 0 state is addressed and 1 if a 1 state is addressed. This can be expressed as $[0] = 0$ and $[1] = 1$, using the notation $[X]$ to represent the value outputted by the RAM when the memory location addressed is in state X .

8.3.3. State transitions in a PLN

The Probabilistic Logic Node (PLN) (e.g., [KanA89], also [Ale89a]) is an extension to the RAM node, in which memory locations can hold the values 0, 1, or u . Similarly to the RAM, a memory location in a PLN can be viewed as being in one of the states 0, 1 or U . When these states are addressed, during the use phase, the corresponding output values are $[0] = 0$, $[1] = 1$ and $[U] = 0$ or 1, chosen randomly with equal probability. For the *train* phase, the equation expressing the state transitions takes a form similar to that of (8.1):

$$S' = S \cdot (\bar{w} + \bar{a} \cdot w) + a \cdot w \cdot \{x \cdot (\bar{s}_0 \cdot 1 + s_0 \cdot U) + \bar{x} \cdot (\bar{s}_1 \cdot 0 + s_1 \cdot U)\}, \quad (8.6)$$

with

$$\begin{cases} s_{X_i} \equiv (S = X_i) \\ \bar{s}_{X_i} \equiv (S \neq X_i). \end{cases} \quad (8.7)$$

These transitions are represented in Figure 8.4.(a), in the case of the write operation.

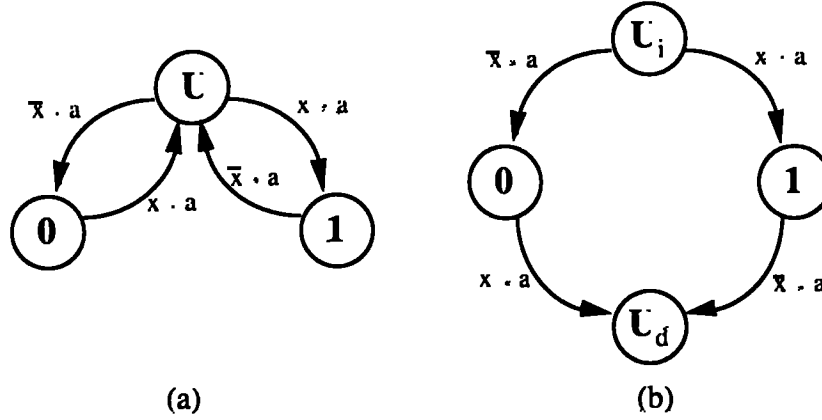


Fig. 8.4: State transition diagram of a memory location in a PLN, during a write operation ($w = 1$). (a) No distinction is made between an initial U state and a disrupted U state. (b) A distinction is made between an initial U state, denoted U_i , and a U state resulting from the disruption of a trained pattern, denoted U_d .

Initially, the state of all memory locations is set to U . The state of an addressed location ($a = 1$) becomes 0 or 1 if the data-in value is 0 or 1 , respectively. If the state of the addressed location ($a = 1$) is 1 and the data-in value is 0 then the state of this location is reset U . State 1 is said to be *disrupted*, or to be in *contradiction* with state 0 . Similarly, if the state of the addressed location is 0 and the data-in value is 1 then the state of this location is reset to U . Otherwise the addressed location remains in the same state. As in the case of the RAM, memory locations not addressed are not affected by the write operation.

It is possible to distinguish between an initial U state, U_i , and a U state resulting from the disruption of a trained pattern, U_d . In that case, the state transition diagram can be represented as in Figure 8.4.(b). The distinction between U_i and U_d may be necessary in the case where it is decided that a memory location that has been disrupted, remains, subsequently, in a U state forever. Then, using (8.4), the update equation (8.6) is transformed into:

$$S' = S \cdot (\bar{w} + \bar{a} \cdot w) + a \cdot w \cdot \left\{ S_{U_d} + x \cdot [(s_1 + s_{U_i}) \cdot 1 + s_0 \cdot U_d] + \bar{x} \cdot [(s_0 + s_{U_i}) \cdot 0 + s_1 \cdot U_d] \right\}. \quad (8.8)$$

The term S_{U_d} can be extracted from the brackets of the second right term of (8.8) by substituting (8.5) in the first term. It holds that:

$$S' = S_{U_d} + S \cdot (\bar{w} + \bar{a} \cdot w) + a \cdot w \cdot \left\{ x \cdot [(s_1 + s_{U_i}) \cdot 1 + s_0 \cdot U_d] + \bar{x} \cdot [(s_0 + s_{U_i}) \cdot 0 + s_1 \cdot U_d] \right\}. \quad (8.9)$$

During a read operation the output values associated with the different states in which the addressed memory location can be are:

$$[0] = 0 \qquad [1] = 1 \qquad [U_i] = [U_d] = 0 / 1$$

8.3.4. State transitions in a GRAM

In a Generalising Random Access Memory (GRAM) the train and use phases are identical to those in a PLN. The additional spreading phase affects memory locations not addressed during the training phase. The spreading phase only affects memory locations which are in state U (or U_i , if a distinction between U_i and U_d is made).

In order to characterise the influence on a memory location, of its neighbours, a variable g is introduced, which takes one of the values 0, 1, or U : 0, if at least one of the neighbouring locations is in the state 0 and the others are in states 0 or U ; 1, if at least one of the neighbouring locations is in the state 1 and the others are in states 1 or U ; U , otherwise. Using the notations (8.3) and

$$g_j \equiv (g = j), \quad (8.10)$$

the state transition of a memory location during the spreading phase can be expressed by:

$$S' = \bar{\sigma} \cdot S + \sigma \cdot \{ (s_1 + g_1 \cdot s_U) \cdot 1 + (s_0 + g_0 \cdot s_U) \cdot 0 + g_U \cdot s_U \cdot U \}, \quad (8.11)$$

σ being a Boolean variable that takes the value 1 during spreading and the value 0 at other times. Equation (8.11) is applied synchronously to all memory locations and is repeated a number Γ of times corresponding to the desired degree of generalisation. The state transitions of a memory location during the spreading phase are shown in Figure 8.5.

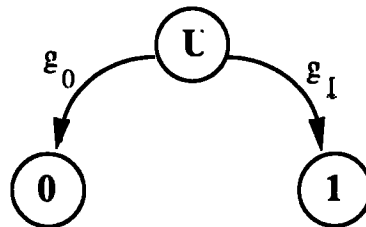


Fig. 8.5; State transition diagram of a memory location in a GRAM, during the spreading phase ($\sigma = 1$).

8.3.5. State transitions in a DGRAM

8.3.5.1. Spreading levels

In order for the DGRAM to learn additional patterns, after previous patterns have spread their states throughout the DGRAM, it is necessary to be able to distinguish between states associated with different *spreading levels*. The state of a memory location is said to be at a spreading level i when the memory location is at a Hamming distance i from the memory location which determined its state through spreading. Therefore the state of a memory location will be labelled by adding a subscript i , whose value is the spreading level of the state.

8.3.5.2. The states of a memory location

A memory location can be in one of the states U , 1_i , 0_i , U_i or R_i , with $i = 0, 1, \dots, N$. U is the initial state of a memory location; its associated output is $[U] = 0/1$. The 1_i are states resulting from the spreading from an addressed memory location trained with a 1 pattern; their associated output value is $[1_i] = 1$. The 0_i are states resulting from the spreading from an addressed memory location trained with a 0 pattern; their associated output value is $[0_i] = 0$. The U_i are states resulting from a contradiction between states 1_i and 0_i ; their associated output value is $[U_i] = 0/1$. Finally, the R_i are states assigned to memory locations from which a pattern is being removed; their associated output value is $[R_i] = 0/1$.

8.3.5.3. Write and remove operations

These operations affect an addressed memory location ($a = 1$) when the external *write* signal w has the value 1. There is an external *remove* signal r , whose value determines whether the operation is write ($r = 0$) or remove ($r = 1$). An addressed memory location in which a pattern 0 or 1 is written, is set to state 0_0 or state 1_0 , respectively. An addressed memory location from which a pattern 0 or 1 is removed is set to state R_0 . The remove operation is necessary in order to be able to completely erase a pattern from the DGRAM.

The next state S'_{ext} of the memory location, resulting from the interaction with the external signals a , x , w and r can then be written:

$$S'_{ext} = \bar{r} \cdot x \cdot (\overline{s_{0_0}} \cdot 1_0 + s_{0_0} \cdot U_0) + \bar{r} \cdot \bar{x} \cdot (\overline{s_{1_0}} \cdot 0_0 + s_{1_0} \cdot U_0) + r \cdot R_0, \quad (8.12)$$

with the notations (8.3), already mentioned, and

$$\overline{s_{x_i}} \equiv (S \neq X_i). \quad (8.13)$$

These state transitions are represented in a diagram in Figure 8.6.

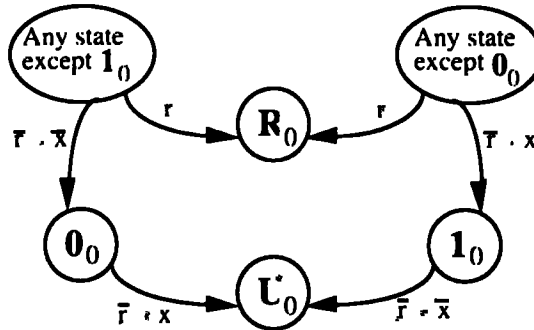


Fig. 8.6: State transition diagram of a memory location in a DGRAM, during write and remove operations ($w = 1$ and $a = 1$).

When a pattern is stored at, or removed from a location, by external addressing, the dynamical system formed by all the memory locations of the DGRAM is no longer in a stable state. It is the interactions between neighbouring memory locations (see next subsection) that enable the system to settle in a new stable state for which the node has the correct generalisation with respect to all the trained patterns.

8.3.5.4. Interactions with neighbouring memory locations

The influence of neighbouring memory locations (at Hamming distance 1) is taken into account by a variable g_k associated with the considered memory location. First, are considered all the memory locations at distance 1 from the location to be updated, whose states have the lowest spreading level. The value of this spreading level is defined as $k - 1$. The states of the chosen memory locations form a set denoted M_{k-1} . The initial state U is defined as having the highest spreading level, say $N + 1$. The variable g takes one of the values $0_k, 1_k, R_k$ or U_k : 0_k , if at least one element of M_{k-1} has the value 0_{k-1} and the others have values 0_{k-1} or U_{k-1} ; 1_k , if at least one element of M_{k-1} has the value 1_{k-1} and the others have values 1_{k-1} or U_{k-1} ; R_k , if at least one element of M_{k-1} has the value R_{k-1} ; and U_k otherwise.

Let the considered memory location be in a state of spreading level i and the value of g , characterising the influence of the neighbours, be of level k . The next state $S'_{d,rr}$ of the memory location, resulting from the dynamical interactions between neighbouring locations, can then be written:

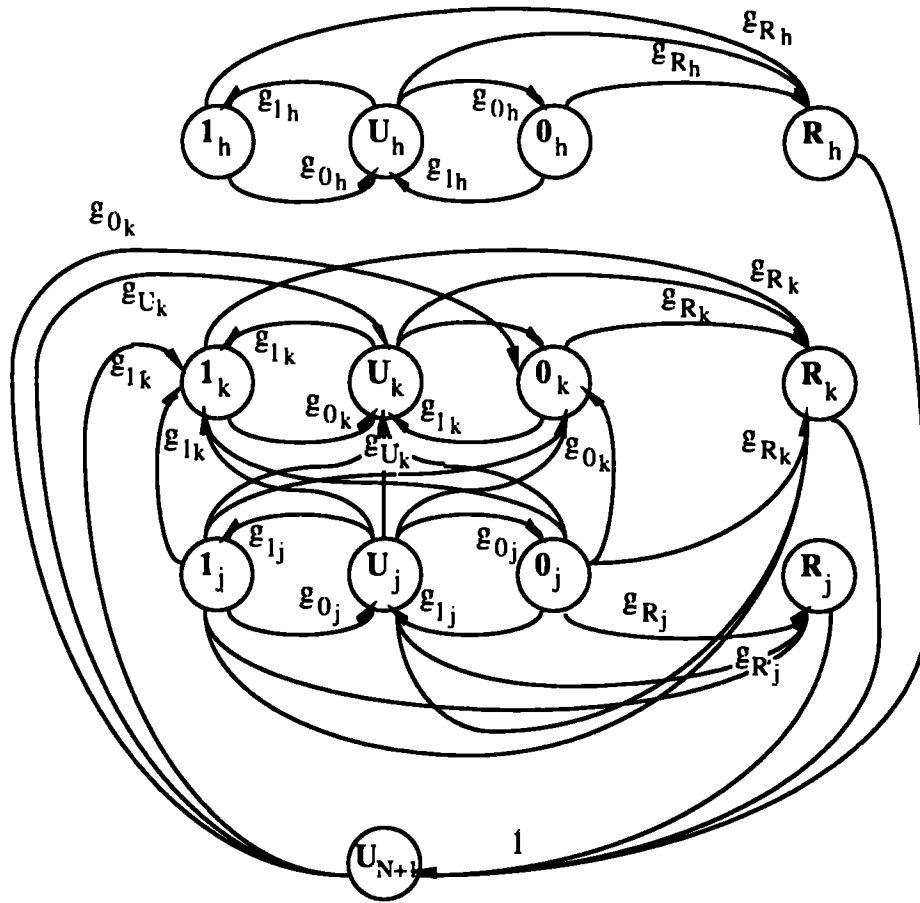


Fig. 8.7; State transition diagram of a memory location in a DGRAM. Only state transitions resulting from the interactions of the memory location with neighbouring locations are shown. The spreading levels h, k, j are such that $0 < h < k < j < N+1$.

$$\begin{aligned}
 \mathbf{S}'_{dyn} = & s_{R_i} \cdot \mathbf{U}_{N+1} \\
 & + (i < k) \cdot \overline{s_{R_i}} \cdot \mathbf{S} \\
 & + (i > k) \cdot \left\{ \overline{s_{R_i}} \cdot (\mathbf{G}_{0_i} + \mathbf{G}_{1_i} + \mathbf{G}_{U_i}) + \overline{s_{U_{N+1}}} \cdot \mathbf{G}_{R_i} + g_{R_i} \cdot \mathbf{S}_{U_{N+1}} \right\} \\
 & + (i = k) \cdot \left\{ (g_{0_i} + g_{U_i}) \cdot \mathbf{S}_{0_i} + (g_{1_i} + g_{U_i}) \cdot \mathbf{S}_{1_i} + g_{U_i} \cdot \mathbf{S}_{U_i} \right. \\
 & \quad \left. + (g_{0_i} \cdot s_{1_i} + g_{1_i} \cdot s_{0_i}) \cdot \mathbf{U}_i \right. \\
 & \quad \left. + s_{U_i} \cdot (\mathbf{G}_{0_i} + \mathbf{G}_{1_i}) + (s_{1_i} + s_{0_i} + s_{U_i}) \cdot \mathbf{G}_{R_i} \right\}
 \end{aligned} \tag{8.14}$$

with the notations

$$g_{X_i} \equiv (g = X_i) \tag{8.15}$$

and

$$\mathbf{G}_{x_i} = g_{x_i} \cdot \mathbf{X}_{j_i} \quad (8.16)$$

and using the notation defined by (8.4). These state transitions described by (8.14) are represented in Figure 8.7, omitting the transitions that do not produce a change of state. In Figure 8.7, the variables h and j correspond to variable i in (8.14), in the cases $i < k$ and $i > k$, respectively,

8.3.5.5. General update equation

The general update equation for a memory location in a DGRAM is obtained by combining the effects of write/remove operations, described by (8.12), with those resulting from the interactions between neighbouring locations, described by (8.14). Thus, it holds that:

$$\mathbf{S}' = (\bar{w} + \bar{a} \cdot w) \cdot \mathbf{S}'_{dyn} + a \cdot w \cdot \mathbf{S}'_{ext} \quad (8.17)$$

8.3.5.6. Stability of the dynamical learning process

The learning process described by (8.12), (8.14) and (8.17) leads to stable states of the network. This can be shown, for the different interactions in the system, by considering the following observations.

For the external addressing, whatever the spreading level of the state of the addressed memory location, this spreading level becomes 0 after the 'storage' of the external pattern.

For the interactions with neighbours which only involve states **1**, **0** and **U**, the next spreading level of the state of a memory location is always the minimum value between the spreading level of the current state of the memory location and the spreading level associated with the variable g characterising the influence of the neighbouring locations.

For the interactions with neighbours which also involve states **R**, If the state of a memory location becomes **R**, whatever its spreading level, this state becomes, unconditionally, $\mathbf{U}_{N+1}$ (or $\mathbf{U}$) at the next time step. Furthermore, this state **R**, whatever its spreading level, 'spreads' to its neighbours if these, whatever their spreading levels, are in states **1**, **0** and **U**. However, in order to guarantee the stability of the process, it is imposed that a **R** state cannot spread to a neighbouring $\mathbf{U}_{N+1}$ (or $\mathbf{U}$) state.

The above definitions and observations guarantee the stability of the learning and removal processes.

8.4. Experimental results

Numerous simulations of the model described in the previous section have been performed, involving experiments of storage and removal of patterns in DGRAM of different sizes. Typical sizes are from $N = 4$ to $N = 20$. Here these experiments are illustrated with an example in the case $N = 5$.

8.4.1 The memory diagram

In order to represent the DGRAM memory space, a variation of the Hasse diagram [Fle71] is introduced, which is a projection of the DGRAM's N -dimensional space onto a 2-dimensional graph preserving the Hamming distance topology. This variation of the Hasse diagram is here called *memory diagram*. It is represented in Figure 8.8 in the case of a 5-input DGRAM.

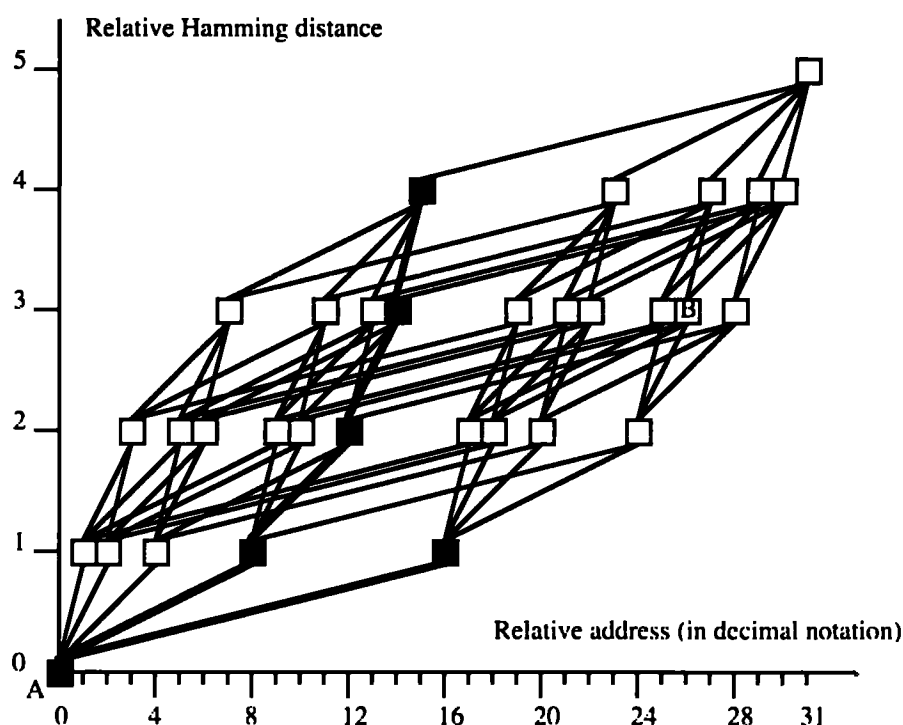


Fig. 8.8: The memory diagram of a 5-input DGRAM. The memory locations are represented as boxes, placed at the vertices of a 5-dimensional cube.

The memory diagram is used here as a graph whose vertices correspond to the memory locations of the DGRAM, and whose axes join memory locations at Hamming distance 1 from each other. A single memory location is chosen arbitrarily as the origin of the graph. Switching between origin locations will then give a varying "representational perspective"

on the DGRAM. The other memory locations are placed at abscissas which correspond to the decimal representation of the location's addresses, relative to the address of the location chosen as the origin. The memory locations at ever increasing Hamming distance from the origin are then drawn in subsequent rows, each row accommodating these at a particular Hamming distance. This has for consequence that each Boolean element of the address of a memory location is associated with an arc orientation on the page.

If location at address A , in decimal representation, is chosen as the origin $(0,0)$, then a location at address B will have graph coordinates (x,y) which satisfy:

$$\begin{cases} x = (A + B) \cdot \text{mod } 2^N \\ y = d_H(A,B), \end{cases}$$

with $d_H(A,B)$ denoting the Hamming distance between A and B . For example, if $A = 20$ (10100) and $B = 14$ (01110), then $x = 26$ and $y = 3$. This is represented in Figure 8.8.

8.4.2. Example 1

Figure 8.9, shows the evolution through time of the contents of characteristic memory locations of a 5-input DGRAM during the storage of 2 patterns, followed by the removal of one of the stored patterns. The locations chosen for the purpose of graphical representation, form a "chain" in which any 2 adjacent elements are at a Hamming distance 1 from each other (see Figure 8.8). The contents of memory locations with addresses 16 (10000), 0 (00000), 8 (01000), 12 (01100), 14 (01110) and 15 (01111) is shown.

Pattern 1 is stored at address 10000, at time T_1 , and spreads to the rest of the DGRAM. At time T_2 , pattern 2 is introduced at address 01100 and, again, spreading takes place. At time T_R , a "removal pattern" is introduced at address 10000 in order to remove pattern 1. This leads to pattern 2 occupying the whole of the memory space, from time T_{R+1} onwards.

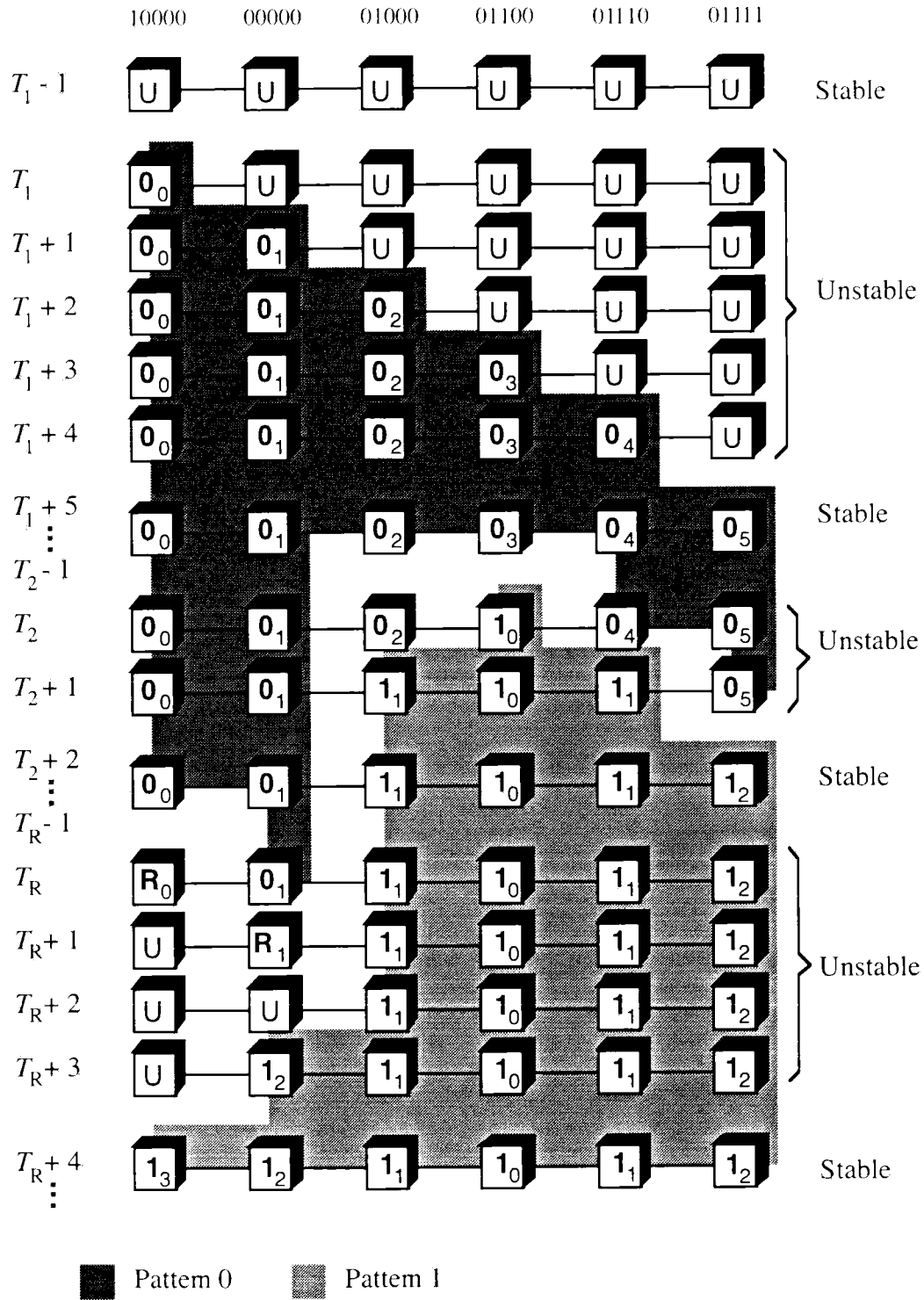


Fig. 8.9: Evolution through time of the contents of characteristic memory locations of a 5-input DGRAM. First two patterns are stored, followed by the removal of one of the stored patterns.

8.4.3. Example 2: Transitory effect

The following example shows the transitory effect occurring when the removal of a pattern (and its spread patterns) is attempted, by addressing one of the spread versions of the pattern with a removal signal. Figure 8.10. shows the evolution through time of the memory location contents.

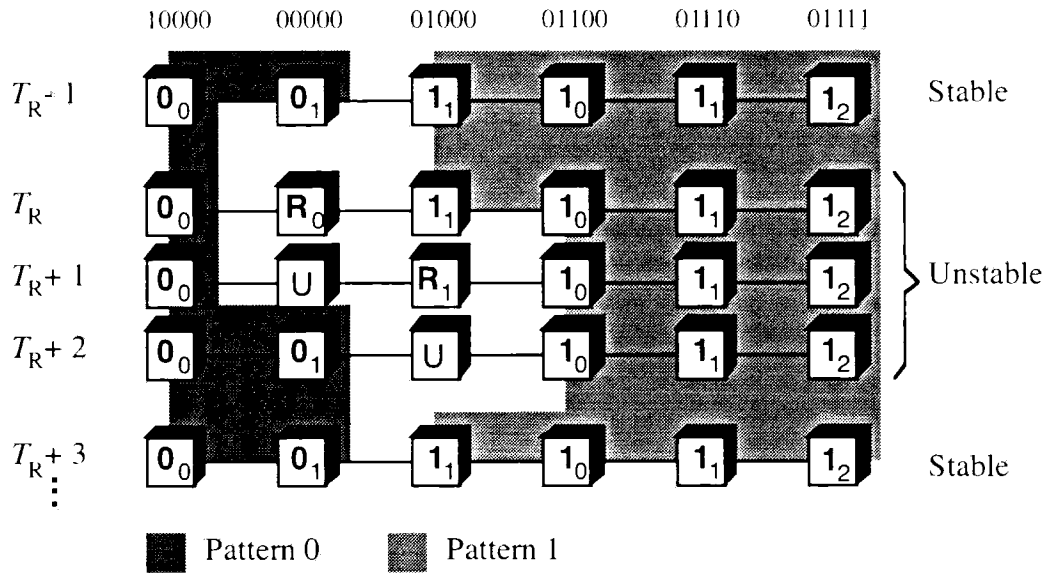


Fig. 8.10: Evolution through time of the contents of characteristic memory locations of a 5-input DGRAM, showing the transitory effect occurring when the removal of a pattern is attempted by addressing one of the spread patterns with a removal signal.

After a short transitory period, the pattern to be removed is restored as previously. It is interesting to note that (8.14) implies that for the removal of a pattern, the original location must be addressed. Otherwise, the removal wave extinguishes after 3 time steps without succeeding in removing pattern 1. It is possible, however, to modify (8.12) and (8.14) to accommodate the possibility of removal of a pattern and its spread patterns, by addressing one of the spread patterns with a removal signal, instead of addressing directly the pattern itself.

8.5. Conclusions

In this chapter, a fundamental viewpoint was adopted in the description of weightless neural nodes, that a memory location contents is no longer considered as the value that is outputted when a memory location is addressed, but as an *internal state* in which a memory location can be. Then, the actual output value of the DGRAM, resulting from the addressing of a memory location, becomes a function of the state of that addressed

location. This viewpoint has led to the definition of the DGRAM.

The DGRAM was derived from the GRAM, in which the training and spreading operations are performed in a single phase. The DGRAM is able to store patterns and spread them, via a dynamical process, governed by equations (8.12) to (8.17) above, involving interactions between each memory location and its neighbours and external signals. The DGRAM exhibits certain advantages over the GRAM. First, it is possible to store additional patterns even after the spreading of previous patterns. Secondly, it is possible to distinguish between trained and spread patterns. And finally it is possible to remove trained patterns and their associated spread patterns without affecting the remaining stored patterns.

Other advantages of the DGRAM include the inherent parallelism in the addressing operation; there is no need for Hamming distance calculations, as it is the case in the V-RAM implementation of the GRAM (see Section 2.3.7.2). Finally, the concept of DGRAM can be extended to allow for multi-valued inputs.

CHAPTER IX. CONCLUSIONS

9.1 Summary of the Thesis Contributions

9.1.1 Mathematical analysis

This thesis has brought together various analyses of weightless artificial neural networks. One of the thesis contributions was an improvement in the understanding of the theoretical properties of WANNs.

The thesis has investigated several WANNs analytically using mathematical tools. These were mainly probability theory, combinatorics, stochastic processes and automata theory. The theoretical results obtained in this thesis, which were mainly concerned with storage capacities and network dynamics, were primarily obtained by methods consisting of counting the number of configurations of the system's parameters which satisfied a number of criteria. This methodology had the essential advantage that it enabled realistic assumptions to be made as to the relative parameter values of the neural models and training algorithms analysed. However, due to the combinatorial explosion of the number of possible configurations of the system's states, often, results could only be obtained for small values of the parameters.

In addition to an extensive review of previous work on WANNs, the other contributions of the thesis were the development of the C-RAM, CDN and DGRAM node models, as well as learning algorithms and spreading procedures for these systems. These contributions are summarised in the next sub-sections.

9.1.2. Review of previous work

Chapters 2 and 3 of the thesis were dedicated to a review of previous work on WANNs. In Chapter 2, the major models of weightless nodes and their properties were reviewed, detailing their memory structure, learning and generalisation properties. This was followed by a review of the most important weightless network architectures and major studies of network behaviours. In Chapter 3, the generality of the weightless approach was argued.

9.1.3. Feed-forward regular pyramidal WANNs

Chapter 4 was concerned with feed-forward pyramidal WANNs. Results about the functional capacity, the storage capacity and the learning dynamics of such networks were

obtained.

An exact and non-recursive formula was derived for the functional capacity of a D -layer pyramidal network with 2-input nodes ($N = 2$), and it was shown that the functional capacity of the network grows as 6^{2D} . An approximation to the functional capacity of a pyramid with general parameters N and D was derived.

A methodology was proposed in order to investigate the storage capacity of the network, defined as the number of arbitrary input-output associations that it can store. An exact probability distribution was obtained, by exhaustive search, for the case $N = 2$ and $D = 2$.

The ASA training algorithm was considered and the learning dynamics of the pyramidal network, trained on the parity checking problem (PCP), were derived. The number of solutions to the PCP was calculated, for general parameters N and D . A calculation of the transition probabilities of the pyramid's internal state, during training of the PCP, was carried out for the case $N = D = 2$. The contents of the network trained under the ASA to the PCP, was proved to converge towards a fully trained configuration.

9.1.4. Unsupervised learning in WANNs

Chapter 5 of this Thesis was concerned with unsupervised learning in WANNs. A self-organising WANN similar in structure and operation to a Kohonen network was presented. The C-discriminator node was introduced and it was shown that such a model can be used in a network governed by an unsupervised training procedure. Simulations have shown that the C-discriminator network is able to form a topologically ordered map of input data, where responses to similar patterns are clustered in certain regions of the output map. It was shown that the output responses of the neurons can be predicted on the basis of the overlap areas between training patterns. The function of experimental work was primarily to validate the theoretical foundations of the training algorithm and not to demonstrate solution of any real-world classification problem.

The main contribution of Chapter 5 was the introduction of a new training algorithm used to train the C-discriminator network, using a spreading mechanism affecting memory locations not addressed by the training patterns. During the training procedure, the information, concerning how a neuron should respond to new patterns, is spread to each memory location of the C-discriminator's C-RAMs. A C-discriminator situated in the excitation region about the firing neuron will become more susceptible to fire when the same or similar input pattern is presented again. Moreover, as a consequence of the above spreading procedure, this C-discriminator will become less susceptible to fire if an input pattern, very different from the one with which the training phase was performed, is presented to the network. Inverse behaviour is observed for a C-discriminator situated in

the inhibition region about the firing neuron.

An additional advantage of the spreading algorithm is that the problem of node saturation, which is a characteristic drawback of n -tuple systems has been eliminated in the C-discriminator network,

Implementation issues were not explicitly addressed. In that respect, it should be noted that the C-discriminator training algorithm, as exposed in Section 5.3.3, is not straightforward to implement in hardware. This is due to the continuous values of the memory contents and the spreading of information to each memory location of the nodes at each step of the training algorithm. However, approximations are possible. The memory contents can be restricted to a range of discrete values in the interval $[0,1]$. Furthermore, the spreading function of the training algorithm can be simplified, such as represented in Figure 5.16. Finally, the training of the network can all together be performed by software and the trained network be subsequently implemented in hardware, using existing state-of-the-art random access memory techniques.

9.1.5. Storage capacity of the GNU network

In Chapter 6, the ability of a weakly interconnected GNU network to perform auto-association was studied. The GNU was therefore operated with the number of external inputs N per node being equal to 0, the number of feedback connections F to a node being different from 0 and equal to some small fraction of the total number K of system outputs.

The disruption probabilities between training patterns in the GNU were calculated and the storage capacity of the network was derived. Two kinds of patterns were considered: first, equally and maximally distant patterns; and then, random patterns. The method of investigation was purely combinatorial, consisting of counting the average number of disruptions between patterns, occurring in the GNU network storing a number S of patterns. The calculations were not based on the retrieval characteristics of the GNU, i.e., whether the network converges towards the nearest stored pattern or drifts away in state space.

It was proved that, when the number S of equally and maximally distant patterns, stored in a weakly interconnected auto-associative GNU network of K nodes, and the number F of inputs per node are asymptotically large, a number $\frac{K}{e^2}$ of nodes exist, on average, in the GNU, in which a particular training pattern is disrupted. Therefore, under the above conditions, the GNU network can store about the same number S of patterns as the number F of inputs per node, with a small error due to memory locations containing u values being addressed and therefore outputting at random. This disruption error means

that, provided a stored pattern is stable, a number $\frac{K}{2e^2}$ of its pixels will, on average, have incorrect values at the terminals of the GNU, resulting in a noisy version of the original stored pattern.

It was proved that, when, in a weakly interconnected auto-associative GNU, the number S of stored patterns, assumed to be *random*, and the number 2^F of memory locations per node become asymptotically large, then a fraction $1 - \frac{1}{\sqrt{e}}$, or 39.35%, of the total number S of patterns is disrupted in any node of the network. Therefore, in a GNU of K nodes, storing 2^F random patterns, a number $K \cdot \left(1 - \frac{1}{\sqrt{e}}\right)$ of nodes exist, on average, in which a particular training pattern is disrupted.

It was proved that a weakly interconnected auto-associative GNU network, with F inputs per node, can store a number $\alpha \cdot 2^F$ of random patterns, where α is a storage ratio which is directly related to the probability $P_d(S, F)$ of disruption between patterns, by the relation $\alpha = 2 \cdot P_d(S, F)$. This means that the greater the fraction of the patterns allowed to be disrupted in the GNU nodes, the more patterns can be stored in the network. However, the probability of disruption needs to remain small in order to enable correct retrieval of the stored patterns.

Finally, a method for improving the immunity of the GNU network to disruptions was proposed, which used a network based on the GNU but in which more than one GRAM code for each GNU output bit. It was proved that the number S of equally and maximally distant patterns that can be stored in a weakly interconnected auto-associative GNU network, with F inputs per node and in which a number M of GRAMs code for each GNU output, is given by $S = M \cdot F$. In the case of random patterns, it was proved that the network can store a number $\alpha \cdot 2^F$ of random patterns, where α is a storage ratio which is directly related to the probability of disruption $P_d(S, F, M)$, by the relation $\alpha = 2 \cdot \sqrt[M]{P_d(S, F, M)}$. The main conclusion remains that using more than one GRAM to code for a GNU output, results in a substantial increase in the storage capacity of the network.

9.1.6. Retrieval process in the GNU network

In Chapter 7, the retrieval process in the GNU network was studied. First, the relationships between pattern overlaps were derived and several useful corollaries were formulated, using the principle of inclusion and exclusion. An important result was that the overlap between a odd number of patterns could be expressed as a linear function of lower order overlaps, whereas no such expression existed in the case of the overlap between an even number of patterns.

The retrieval equations of the GNU network were established in the case of three arbitrary stored patterns. These equations govern the dynamics of the system. The main contribution was to show that the retrieval process in a GNU, even for 3 arbitrary stored patterns, leads to complicated equations between pattern overlaps.

9.1.7. Dynamical spreading

In Chapter 8, a fundamental viewpoint was adopted in the description of weightless neural nodes. The contents of a memory location was no longer considered as the value which is outputted when a memory location is addressed, but as an *internal state* which a memory location can be in. Then, the actual value outputted by the node, as a result of the addressing of a memory location, becomes a function of the state of the addressed location. This viewpoint has led to the definition of the DGRAM model.

The DGRAM was derived from the GRAM, in which the training and spreading operations are performed in a single phase. The DGRAM is able to store patterns and spread them, via a dynamical process, involving interactions between each memory location and its neighbours and external signals. The equations governing this dynamical process were derived and the learning process was shown to lead to stable states of the memory contents.

The DGRAM exhibits certain advantages over the GRAM. First, it is possible to store additional patterns even after the spreading of previous patterns. Secondly, it is possible to distinguish between trained and spread patterns. And finally it is possible to remove trained patterns and their associated spread patterns without affecting the remaining stored patterns.

9.2. Suggestions for future work

The various analyses carried out in this thesis have focused on a series of specific weightless models, with specific topologies and often restricted sets of parameter values. Answers have been given to a restricted set of questions concerning the storage capacity, training algorithms and learning dynamics of these systems. Many more aspects of these systems, both theoretical and experimental, could be investigated. In the following paragraphs, a selected set of problems is outlined, which relate to further aspects the theory of WANNs and which would be worth investigating as a continuation to the work presented in this thesis.

The spreading function used in the C-discriminator network, which was developed in Chapter 5 of this thesis, can be approximated to functions which would only affect a restricted number of memory locations in the C-discriminator's C-RAMs. A very coarse approximation is the one obtained when only two memory locations are affected at each training step: the addressed location and either the opposite location or some other location

at some specific Hamming distance from the addressed location. It would be interesting to study how these approximations affect the performance of the system.

The simulations of the C-discriminator network, presented in Chapter 5 of this thesis, were extremely limited in size and the training data were intentionally chosen to be tiny. Indeed, the purpose of these experiments was primarily to validate the theoretical foundations of the C-discriminator training algorithm and not to demonstrate solution of any real-world classification problem. It would, however, be extremely useful to investigate how the system performs on large real-world training data sets. Also important would be a performance comparison to networks such as the standard Kohonen network or the Allinson network.

The GNU network, whose storage properties were investigated in Chapter 6 and retrieval equations derived in Chapter 7, was (1) weakly interconnected ($F \ll K$), (2) auto-associative ($N = 0$) and (3) autonomous. These 3 restrictive operating conditions immediately suggest 3 corresponding extensions to the network analysis:

- (1) The storage and retrieval properties of the GNU could be studied in the case when the degree of feed-back, F , can no longer be considered as a very small fraction of the number K of nodes in the network.
- (2) The properties of a GNU network with external inputs could be investigated. In this level-2 structure (see Section 2.5.1), the input field to the network is split between an external input field and a feed-back input field. It has been shown experimentally that such a structure can be trained to respond to a static stimulus, on its input field, by a static response or by a sequence of responses, on its feed-back terminal. Correspondingly, the network can also be trained to respond to a sequence of different stimuli by a static response or by a sequence of responses.
- (3) Systems consisting of several interconnected GNU networks could be studied and simulated. In [AleM93], such neural systems were proposed in order to build modular cognitive systems, exhibiting advanced state structures. However, theoretical aspects of these systems, such as the design of overall training procedures and interconnection schemes, remain, to date, untouched and would therefore be worth investigating.

In this thesis, several spreading functions were considered, which were associated with learning in nodes such as the GRAM, the CDN and the DGRAM. These spreading functions were all dependent on the Hamming distance between the memory location addressed by a training pattern and a memory location to be 'spread'. An obvious extension is to allow for spreading functions based on different distance metrics. In a decomposed structure such as the discriminator node, an exponential metric is obtained (see Section 2.3.3.2); this distance metric is due to the topological structure of the node and not to an explicit spreading of information throughout the memory locations of the node. It would be useful to design spreading procedures implementing several distance metrics other than the

Hamming distance metric. Indeed, the primary aim, when the concept of spreading was introduced [Ale90a], was to allow the neural network designer to shape the generalisation properties of the WANN in order to achieve some desired system behaviour.

REFERENCES

- [A-Ala90] Al-Alawi, R., "The Functionality, Training and Topological Constraints of Digital Neural Networks", Ph.D. Thesis, Brunel University, Oct. 1990.
- [A-Ala89] Al-Alawi, R., and Stonham, T. J., "Functionality of Multilayer Boolean Neural Networks", *Electronics Letters*, **25**, 10, pp. 657-8, May 1989.
- [AckHS85] Ackley, D. H., Hinton, G. E., and Sejnowski, T. J., "A Learning Algorithm for Boltzmann Machines", *Cognitive Science*, **9**, pp. 147-69, 1985.
- [Alb75a] Albus, J. S., "A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)", *Journal of Dynamic Systems, Measurement, and Control*, Trans. ASME, Series G, **97**, 3, pp. 220-7, Sept. 1975.
- [Alb75b] Albus, J. S., "Data Storage in the Cerebellar Model Articulation Controller (CMAC)", *Journal of Dynamic Systems, Measurement, and Control*, Trans. ASME, Series G, **97**, 3, pp. 228-33, Sept. 1975.
- [Ale65] Aleksander, I., "Fused Logic Element which learns by Example", *Electronics Letters*, **1**, 6, pp. 173-4, August 1965.
- [Ale70] Aleksander, I., "Brain Cell to Microcircuit", *Electronics and Power*, February 1970.
- [Ale73] Aleksander, I., "Random Logic Nets: Stability and Adaptation", *Int. Journal of Man-Machine Studies*, **5**, pp.115-31, 1973.
- [Ale83a] Aleksander, I., "The Analysis of Digital Neural Nets", *Int. Rep.*, Dept. Elec. Eng., Brunel University, England, 1983.
- [Ale83b] Aleksander, I., "Emergent Intelligent Properties of Progressively Structured Pattern Recognition Nets", *Pattern Recognition Letters*, **1**, 5-6, pp. 375-84, 1983.
- [Ale84a] Aleksander, I., "WISARD: A Component for Image Understanding Architectures", *Artificial Vision for Robots* (Ed. Aleksander), Chapman & Hall, New York, 1984.
- [Ale84b] Aleksander, I., "Memory Networks for Practical Vision Systems: Design Calculations", Ch.11, in *Artificial Vision for Robots* (Ed. Aleksander), Chapman & Hall, New York, 1984.
- [Ale86] Aleksander, I., "The Generalised Implementation of Associative Networks for Vision Systems", *Int. Rep.*, Alvey Vision Club, Dept. Computing,

- Imperial College, London, 1986.
- [Ale87] Aleksander, I., "Adaptive Pattern Recognition Systems and Boltzmann Machines: A Rapprochement", *Pattern Recognition Letters*, **6**, pp. 113-20, 1987.
- [Ale88] Aleksander, I., "Are Special Chips Necessary for Neural Computing?", *Proc. Int. Workshop on VLSI for Artificial Intelligence*, University of Oxford, 20-22 July 1988.
- [Ale89a] Aleksander, I., "The Logic of Connectionist Systems", in *Neural Computing Architectures*, Ed. I. Aleksander, Kogan Page, London, MIT Press, Boston, 1989.
- [Ale89b] Aleksander, I., "Canonical Neural Nets Based on Logic Nodes", *Proc. 1st IEE Conf. on ANNs*, pp.100-14, London, Oct. 1989.
- [Ale90a] Aleksander, I., "Ideal Neurons for Neural Computers", *Proc. Int. Conf. on Parallel Processing in Neural Systems and Computers*, Düsseldorf, Springer Verlag, 1990.
- [Ale90b] Aleksander, I., "Neural Systems Engineering: Towards a Unified Design Discipline?", *IEE Computing & Control Eng. J.*, **1**, pp. 259-65, 1990.
- [Ale90c] Aleksander, I., "Weightless Neural Tools: Towards Cognitive Macrostructures", *The CAIP Neural Network Workshop*, Rutgers University, New Jersey, October 1990.
- [AleEP93] Aleksander, I., Evans, R., and Penny, W., "Magnus: An Iconic Neural State Machine", *Proc. Weightless Neural Network Workshop'93*, pp. 156-9, University of York, April 1993.
- [AleH76] Aleksander, I., and Hanna, F. K., *Automata Theory: An Engineering Approach*, Edward Arnold (Publ.) Ltd., London, 1976.
- [AleM90] Aleksander, I., and Morton, H. B., "The Secrets of the WISARD", Ch. 5, in *An Introduction to Neural Computing*, Chapman and Hall, London, 1990.
- [AleM91] Aleksander, I., and Morton, H. B., "General Neural Unit: Retrieval Performance", *Electronics Letters*, **27**, 19, pp. 1776-7, Sep. 1991.
- [AleM93] Aleksander, I., and Morton, H. B., *Symbols and Neurons: the Stuff That Mind is made of*, Chapman and Hall, London, 1993.
- [AleS79] Aleksander, I., and Stonham, T. J., "Guide to Pattern Recognition Using Random-Access Memories", *IEE Journal of Computers and Digital Techniques*, **2** (1), pp. 29-40, Feb. 1979.
- [AleTB84] Aleksander, I., Thomas, W. V., and Bowden, P. A., "WISARD, a radical step forward in image recognition", *Sensor Review*, **4**, 3, pp. 120-124, 1984.
- [AleW85] Aleksander, I., and Wilson, M. J. D., "Adaptive Windows for Image

- Processing", *IEE Proc.*, **132**, 5, 1985.
- [AllBJ89] Allinson, N.M., Brown, M.T., and Johnson, M.J., "{0,1}ⁿ-space Self-organising Feature Maps - Extensions and Hardware Implementation", *Proc. 1st IEE Conf. on ANNs*, pp.261-4, London, Oct. 1989.
- [AllJ89] Allinson, N.M., and Johnson, M.J., "Realisation of Self-organising Neural Maps in {0,1}ⁿ-space", in *New Developments in Neural Computing*, Ed. by J. G. Taylor and C. L. T. Mannion, IOP Publishing Ltd, pp. 79-86, 1989.
- [AllJM89] Allinson, N.M., Johnson, M.J., and Moon, K.J., "Digital Realisation of Self-organising Maps", in *Advances in Neural Information Processing Systems I*, Ed. D. S. Touretzky, pp. 728-38, Morgan Kaufmann, San Mateo, California, 1989.
- [AllJ92] Allinson, N.M., and Johnson, M.J., "Self-Organising Topographical Classifiers in {0,1}ⁿ Space", Preprint, Dept. Electronics, University of York, England, 1992.
- [AllK93] Allinson, N.M., and Kolcz, A.R., "Enhanced *N*-tuple Approximators", *Proc. Weightless Neural Network Workshop'93*, pp. 38-45, University of York, April 1993.
- [Ama89] Amari, S.-I., "Characteristics of Sparsely Encoded Associative Memory", *Neural Networks*, **2**, pp. 451-7, 1989.
- [Aus86] Austin, J., "The Design and Application of Associative Memories for Scene Analysis", Ph. D. Thesis, Brunel University, Aug. 1986.
- [Aus88] Austin, J., "Grey Scale *N*-tuple Processing", *Proc. BPRA 4th International Conf. on Pattern Recognition*, Cambridge, pp. 28-30, March 1988.
- [Aus89] Austin, J., "ADAM: An Associative Neural Architecture for Invariant Pattern Classification", *Proc. IEE 1st Int. Conf. on ANNs*, pp.196-200, London, 1989.
- [Aus93] Austin, J., "A Review of the Advanced Distributed Associative Memory", *Proc. Weightless Neural Network Workshop'93*, pp. 24-8, University of York, April 1993.
- [AusS87] Austin, J., "An Associative Memory for Use in Image Recognition and Occlusion Analysis", *Image and Vision Computing*, **5**, 4, pp. 251-61, Nov. 1987.
- [BarSA83] Barto, A. G., Sutton, R. S., and Anderson, C. W., "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems, *IEEE Trans. on Systems, Man and Cybernetics*, **13**, pp. 834-46, 1983.
- [BisFF89] Bisset, D.L., Filho, E., and Fairhurst, M.C., "A Comparative Study of Neural Network Structures for Practical Application in a Pattern Recognition Environment", *Proc. IEE 1st Int. Conf. on ANNs*, pp.378-

- 82, London, Oct. 1989.
- [BleB59] Bledsoe, W. W., and Browning, I., "Pattern Recognition and Reading by Machine", *Proc. Eastern Joint Computer Conf.*, Boston, Mass., 1959.
- [BleB62] Bledsoe, W. W., and Blisson, C.L., "Improved Memory Matrices for the *N*-tuple Pattern Recognition Method", *IRE Trans. on Electronic Computers*, pp.414-5, 1962.
- [Cai89] Caianiello, E. R., "A Theory of Neural Networks", in *Neural Computing Architectures*, Ed. I. Aleksander, Kogan Page, London, MIT Press, Boston, 1989.
- [CamSW89] Campbell, C., Sherrington, D., and Wong, K.Y.M., "Statistical Mechanics and Neural Networks", Ch. 12, in *Neural Computing Architectures*, Ed. I. Aleksander, Kogan Page, London, MIT Press, Boston, 1989.
- [CarFBF91] Carvalho, A., Fairhurst, M.C., Bisset, D.L., and E. Filho, "An Analysis of Self-organising Networks Based on Goal-seeking Neurons", *Proc. 2nd IEE Int. Conf. on Neural Networks*, pp. 257-61, Bournemouth, Nov. 1991.
- [CarG89] Carpenter, G. A., and Grossberg, S., "Search Mechanisms for Adaptive Resonance Theory (ART) Architectures", *Proc. IJCNN'89*, 1, 201-5, Washington D.C., June 1989.
- [CarG90] Carpenter, G. A., and Grossberg, S., "Adaptive Resonance Theory: Neural Network Architectures for Self-organising Pattern Recognition", in *Parallel Processing in Neural Systems and Computers*, R. Eckmiller, G. Hartmann and G. Hauske (Eds.), Elsevier Sciences Publishers B.V.(North Holland), 1990.
- [CarL89] Carroll, J., and Long, D., *Theory of Finite Automata, with an Introduction to Formal Languages*, Prentice-Hall International Inc., 1989.
- [CarP87a] Carnavali, P., and Patarnello, S., "Learning Networks of Neurons with Boolean Logic", *Europhysics Letters*, 4, 4, pp.503-8, 1987.
- [CarP87b] Carnavali, P., and Patarnello, S., "Exhaustive Thermodynamical Analysis of Boolean Learning Networks", *Europhysics Letters*, 4, 10, pp.1199-204, 1987.
- [Cha83] Changeux, J.-P., *L'Homme Neuronal*, Ed. Fayard, Paris, 1983.
- [Cho88] Chou, P. A., "The Capacity of the Kanerva Associative Memory is Exponential", in *Neural Information Processing Systems 2* (Ed. Danna Z. Anderson), pp. 184-91, American Institute of Physics, New York, 1988.
- [Cho89] Chou, P. A., "The Capacity of the Kanerva Associative Memory", *IEEE Trans. on Information Theory*, IT-35. 2, pp. 281-98, 1989.
- [ClaGT89] Clarkson, T. G., Gorse, D. E., and Taylor, J. G., "Hardware Realisable Models of Neural Processing", *Proc. 1st IEE Int. Conf. on ANNs*, pp.242-6, London, 1989.

- [ComRS86] Computer Recognition Systems Ltd., "Wisard System User Guide", Wokingham, Berkshire, 1986.
- [CoxM65] Cox, D. R. and Miller, H. D., *The Theory of Stochastic Processes*, Chapman and Hall, London, New York, 1965.
- [Cul71] Cull, P., "Linear Analysis of Switching Nets", *Kybernetik*, **1**, pp. 31-9, 1971.
- [Den87] Denker, J. & al., "Large Automatic Learning, Rule Extraction and Generalisation", *Complex Systems*, **1**, pp. 877-922, 1987.
- [DerGZ87] Derrida, B., Gardner, E., and Zippelius, A., *Europhysics Letters*, **4**, 167, 1987.
- [Dow75] Dawson, C., *Simple Scene Analysis Using Digital Learning Nets*, Thesis, University of Kent, 1975.
- [FilBF90] Filho, E.C.D.B.C., Bisset, D.L., and Fairhurst, M.C., "A Goal Seeking Neuron for Boolean Neural Networks", *Proc. INNC-90*, **2**, pp. 894-7, Paris, July 1990.
- [FilFB92] Filho, E.C.D.B.C., Fairhurst, M.C., and Bisset, D.L., "Analysis of Saturation Problem in RAM-based Neural Networks", *Electronics Letters*, **28**, 4, Feb. 1992.
- [Fle71] Flegg, H. G., *Boolean Algebra and its Application*, Blackie & Son Ltd., London and Glasgow, 1971.
- [Flo64] Florine, J., *The Design of Logical Machines*, translated and edited by A. R. Cownie and E. M. Hynes, London, English Universities Press, 1973.
- [FogGW82] Fogelman-Soulie, F., Goles-Chacc, E., and Weisbuch, G., "Specific Roles of the Different Boolean Mappings in Random Networks", *Bulletin of Mathematical Biology*, **44**, 5, 715-30, 1982.
- [Fon88] Fong, A. M., "Some Properties of Logical Functions of PLN Networks", *Int. Rep. NSEIR/AMF#1/88*, Imperial College, London, January 1989.
- [Ful93] Fulcher, E. P., Ph. D. Thesis, Imperial College, University of London, 1993.
- [GorT88] Gorse, D. E., and Taylor, J. G., "On the Equivalence and Properties of Noisy Neural and Probabilistic RAM Nets", *Physics Letters A*, **131**, 6, pp. 326-32, 1988.
- [GorT90] Gorse, D. E., and Taylor, J. G., "Training Strategies for Probabilistic RAMs", in *Parallel Processing in Neural Systems and Computers*, R. Eckmiller, G. Hartmann and G. Hauske (Eds.), Elsevier Sciences Publishers B.V.(North Holland), 1990.
- [GorT93] Gorse, D. E., and Taylor, J. G., "A Review of the Theory of P-RAMs", *Proc. Weightless Neural Network Workshop'93*, pp. 13-7, University of York, April 1993.

- [Gre86] Green, D., *Modern Logic Design*, Addison-Wesley, 1986.
- [GriS82] Grimmett, G. R., and Stirzaker, D. R., *Probability and Random Processes*, Clarendon Press, Oxford, 1982.
- [Gur89] Gurney, K., *Learning in Networks of Structured Hypercubes*, Ph.D. Thesis, Dept. Electrical Eng. & Electronics, Brunel University, 1989.
- [Heb49] Hebb, D. O., *The Organisation of Behaviour*, New York, Wiley, 1949.
- [Hop82] Hopfield, J. J., "Neural Networks and Physical Systems with Emergent Collective Computational Capabilities", *Proc. Nat. Acad. Sci. USA*, 79, 2554-58, 1982.
- [HopU79] Hopcroft, J. E., and Ullman, J. D., *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley (Publ.) Inc., 1979.
- [Jor88] Jordan, M., in *Proc. of 1st Connectionist Summer School*, 1988.
- [Jud90] Judd, J. S., *Neural Network Design and the Complexity of Learning*, MIT Press, 1990.
- [Kan86] Kan, W. K., "Multi-layer Associative Networks: Massively Parallel Architectures for AI Applications", *Int. Rep.*, Dept. Computing, Imperial College, London, 1986.
- [Kan88] Kanerva, P., *Sparse Distributed Memory*, MIT Press, 1988.
- [KanA89] Kan, W. K., and Aleksander, I., "A Probabilistic Logic Neuron Network for Associative Learning", in *Neural Computing Architectures*, Ed. I. Aleksander, Kogan Page, London, MIT Press, Boston, 1989.
- [Kau69] Kauffman, S., "Metabolic Stability and Epigenesis in Randomly Constructed Genetic Nets", *Journal of Theoretical Biology*, 22, pp. 437-67, 1969.
- [KerS90] Kerin, M.A., and Stonham, T. J., "Face Recognition using a Digital Neural Network with Self-organising Capabilities", *Proc. 10th Int. Conf. on Pattern Recognition*, pp. 738-41, 1990.
- [Koh82a] Kohonen, T., "Clustering, Taxonomy, and Topological Maps of Patterns", *IJCPR-82*, vol. 6, pp. 114-128, 1982.
- [Koh82b] Kohonen, T., "Self-Organized Formation of Topologically Correct Feature Maps", *Biological Cybernetics*, vol. 43, pp. 59-69, 1982.
- [Koh82c] Kohonen, T., "Analysis of a Simple Self-Organizing Process", *Biological Cybernetics*, vol. 44, pp. 135-140, 1982.
- [Koh88a] Kohonen, T., "An Introduction to Neural Computing", *Neural Networks*, 1, 1, pp. 3-16, 1988.
- [Koh88b] Kohonen, T., "The 'Neural' Phonetic Typewriter", *Computer*, March 1988.
- [Koh89a] Kohonen, T., *Self-Organization and Associative Memory*, 3th edition, Springer-Verlag, New York, 1989.

- [Koh89b] Kohonen, T., "Speech recognition based on topology-preserving neural maps", *Neural Computing Architectures*, Ch. 3, Ed. I, Aleksander, Kogan Page, 1989.
- [Kru91] Kruijswijk, S. G., "Improving Neural Networks by Controlling the Distribution of the Stochastic Transfer Function", Private communication. Contact address: Oud Ehrenstein 1, 1082 AH Amsterdam, The Netherlands, 1991.
- [Luc91] Lucy, J., "Perfect Autoassociators using RAM-type nodes", *Electronics Letters*, **27**, 10, pp. 799-800, May 1991.
- [Mar89] Martland, D., "Adaptation of Boolean Networks Using Back-error Propagation", *Proc. IJCNN-89*, Washington D.C., Dec. 1989.
- [MarA93] Martins, W. and Allinson, N. M., "Two Improvements for GSN Neural Networks", *Proc. Weightless Neural Network Workshop'93*, pp. 58-63, University of York, April 1993.
- [McCulP43] McCulloch, W.S., and Pitts, W.H., "A Logical Calculus of the Ideas Immanent in Nervous Activity", in *The Bulletin of Mathematical Biophysics*, **5**:115-33, Ed. N. Rashevsky, 1943.
- [McEPRV87] McEliece, R. J., Posner, E.C., Rodemich, E. R., and Venkatesh, S. S., "The Capacity of the Hopfield Associative Memory", *IEEE Trans. on Information Theory*, **IT-33**, 461-82, 1987.
- [MinP69] Minsky, M. L., and Papert, S. A., *Perceptrons: An Introduction to Computational Geometry*, MIT Press, Cambridge, Massachusetts, 1969.
- [Moo89] Moore, A. W., "Efficient Memory-based Learning for Robot Control", Ph.D.Thesis, *Technical Report No. 209*, Computer Laboratory, University of Cambridge, 1990.
- [Mrs93] Mrsic-Flögel, J., *Aspects of Planning with Neural Systems*, Ph.D. Thesis, Imperial College, University of London, 1993.
- [Mur65] Muroga, S., "Lower Bounds of the Number of Threshold Functions and a Maximum Weight", *IEEE Trans. on Electronic Computers*, pp. 136-48, 1965.
- [Myc88a] Myers, C.E., "The Number of Functions computed by PLN Trees", *Int. Rep. NSEIR/CM#1/88*, Dept. Elec. Eng., Imperial College, London, March 1988.
- [Myc88b] Myers, C.E., "Learning Algorithms for Probabilistic Neural Nets", *Int. Rep. NSEIR/CM#2/88*, Dept. Elec. Eng., Imperial College, London, 1988.
- [Myc89] Myers, C.E., "Output Functions for Probabilistic Logic Nodes", *Proc. 1st IEE Int. Conf. on Artificial Neural Networks*, pp. 310-4, London, Oct. 1989.
- [Myc90] Myers, C.E., "Learning with Delayed Reinforcement in an Exploratory

- Probabilistic Logic Neural Network", Ph.D. Thesis, University of London, Dec. 1990.
- [Nto90] Ntourntoufis, P., "Self-organization Properties of a Discriminator-based Neural Network", *Proc. IJCNN-90*, 2: 319-24, San Diego, 1990.
- [Nto91] Ntourntoufis, P., "Storage Capacity and Retrieval Properties of an Auto-associative General Neural Unit", *Proc. IJCNN-91*, II, pp. A-959, Seattle, 1991.
- [Nto92a] Ntourntoufis, P., "Non-recursive Formula for the Functional Capacity of a Pyramidal Feed-forward Boolean Neural Network with D Layers of 2-input Nodes", *Int. Rep. NSEIR/PN#1/92*, Dept. Elec. Eng., Imperial College, London, Feb. 1992.
- [Nto92b] Ntourntoufis, P., "Parallel Hardware Implementation of the General Neural Unit Model", *Proc. ICANN-92*, Brighton, 1992.
- [Nto92c] Ntourntoufis, P., "Storage Capacity of a Pyramidal Feed-forward Boolean Neural Network", *Int. Rep. NSEIR/PN#2/92*, Dept. Elec. Eng., Imperial College, London, March 1992.
- [Nto92d] Ntourntoufis, P., "The Dynamically Generalising Random Access Memory", *Int. Rep. NSEIR/PN#3/92*, Dept. Elec. Eng., Imperial College, London, April 1992.
- [Nto93] Ntourntoufis, P., "A Dynamically Generalising Weightless Neural Element", in *Proc. ICANN'93*, pp. 658-61, Amsterdam, Sept. 1993.
- [NtoS94] Ntourntoufis, P., and Stonham, T. J., "A Fast Zooming Mechanism for a Magnus Weightless Neural System", in Press, Sept. 1994.
- [PenGS91] Penny, W.D., Gurney, K.N., and Stonham, T. J., "Reward-Penalty Training for Logical Neural Networks", preprint, Dept. Electrical Eng. & Electronics, Brunel University, England, 1991.
- [PenS91] Penny, W.D., and Stonham, T. J., "Learning Algorithms for Logical Neural Networks", preprint, Dept. Electrical Eng. & Electronics, Brunel University, England, 1991.
- [PenS92] Penny, W.D., and Stonham, T. J., "On the Generalisation Ability and Storage Capacity of Logical Neural Networks", preprint, Dept. Electrical Eng. & Electronics, Brunel University, England, 1992.
- [PenS93] Penny, W.D., and Stonham, T. J., "Storage Capacity of Multi-layer Boolean Neural Networks", *Electronics Letters*, Feb. 1993.
- [PolTW83] Pólya, G., Tarjan, R. E., and Woods, D. R., *Notes on Introductory Combinatorics*, 'Progress in Computer Science' Series, 4, Birkhäuser, Boston, 1983.
- [Red88] Redgers, A., "The Functionality of a Pyramidal Net of Boolean Nodes", *Int. Rep. NSEIR/AR#1/88*, Dept. Elec. Eng., Imperial College, London,

- Dece. 1988.
- [Ree73] Reeves, A. P., *Tracking Experiments with an Adaptive Logic System*, Ph.D. Thesis, University of Kent, 1973.
- [RicS93] Rickman, R. and Stonham, J., "A Novel Learning Strategy for Self-organising Weightless Neural Networks", *Proc. Weightless Neural Network Workshop'93*, pp. 82-6, University of York, April 1993.
- [RumHW86] Rumelhart, D. E., Hinton, G. E., and Williams, R. J., "Learning internal representations by error propagation", in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol.1: Foundations.*, Ch. 8, Ed. D.E. Rumelhart and J.L. McClelland, MIT Press, Cambridge, MA., 1986.
- [RumM86] Rumelhart, D. E., and McClelland, J. L., (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol.1: Foundations.*, MIT Press, Cambridge, MA., 1986.
- [Sha89a] Shapiro, J.L., "Dynamics of Learning in a Neural Network with Hidden Units", Preprint, Dept. Theoretical Physics, University of Manchester, 1989.
- [Sha89b] Shapiro, J.L., "Hard Learning in Boolean Neural Networks", in *New Developments in Neural Computing*, Ed. by J. G. Taylor and C. L. T. Mannion, IOP Publishing Ltd, pp. 125-32, 1989.
- [Ste62] Steck, G.P., "Stochastic Model for the Browning-Bledsoe Pattern Recognition Scheme", *IRE Transactions on Electronic Computers*, pp. 274-82, April 1962.
- [TamS92] Tambouratzis, G., and Stonham, T. J., "Implementing Hard Self-organisation Tasks Using Logical Neural Networks", *Proc. ICANN-92*, pp. 643-6, Brighton, 1992.
- [TatFL89] Tattershall, G. D., Foster, S., and Linford, P., "Single Layer Look-Up Perceptrons", *Proc. 1st IEE Int. Conf. on ANNs*, pp. 148-152, London, 1989.
- [Tho79] Thomas, R. , "Kinetic logic: a Boolean approach to the analysis of a complex regulatory systems", in *Lecture notes in Biomathematics*, Ed. R. Thomas, 29, Springer-Verlag, Berlin, 1979.
- [Ull73] Ullmann, J. R., *Pattern recognition techniques*, Butterworth & Co (Publ.), London, 1973.
- [VdBroK90] Van den Broeck, C., and Kawai, R., "Learning in Feed-forward Boolean Networks", *Physical Review A*, 42, 10, pp. 6210-18, 1990.
- [Vid88] Vidal, J. J., "Implementing Neural Nets with Programmable Logic", *IEEE Trans. Acoustics, Speech & Signal Proc.*, 36, 7, 1180-90, 1988.
- [V-Ham75] Van Ham, Ph., "Discrete Models with Delayed Action", Doctoral Thesis (in

- French), Free University of Brussels, 1975.
- [V-Ham84] Van Ham, Ph., "Asynchronous Logical Models: Kinetic Logic", *Int. Rep. PVH/GP84-0120* (in French), Mathematical Modelisation Cell, Industrial Research Center, Free University of Brussels, 1984.
- [Was89] Wasserman, P. D., "Adaptive Resonance Theory", in *Neural Computing: Theory and Practice*, Chapter 8, pp. 127-49, Van Nostrand Reinhold, 1989.
- [WidH60] Widrow, B., and Hoff, M. E., "Adaptive Switching Circuits", *IRE Western Electronic Show and Convention*, 4, 96-104, 1960.
- [WonS88] Wong, K. Y. M., and Sherrington, D., "Storage Properties of Boolean Neural Networks: the Aleksander Model", *Europhysics Letters*, March 1988.
- [WonS89] Wong, K. Y. M., and Sherrington, D., "Theory of Associative Memory in Randomly Connected Boolean Neural Networks", *J. Phys. A: Math. Gen.*, 22, pp. 2233-63, 1989.
- [WonS91] Wong, K. Y. M., and Sherrington, D., "Statistical Mechanics of Neural Networks: what are the differences between wide and narrow basins?", Preprint, Dept. Theor. Physics, University of Oxford, 1991.
- [ZhaC90] Zhang, S.-W., and Constantinides, A. G., "Lagrange Programming Neural Networks", Preprint, Imperial College, London, 1990.
- [ZhaC91] Zhang, S.-W., and Constantinides, A. G., "Image Restoration Using Lagrange Programming Neural Networks", Preprint, Imperial College, London, 1990.
- [ZhaS91] Zhang, S.-W., and Stonham, T. J., "Pattern Association with Intelligent Memories", *Report IED 3/1005*, Dept. Electrical Eng. & Electronics, Brunel University, England, Oct. 1991.
- [ZhaZZ91] Zhang, B., Zhang, L. and Zhang, H., "The Quantitative Description of PLN Network's Behaviour", in *Artificial Neural Networks I*, Ed. T. Kohonen & al., 2, pp. 1015-8, Elsevier Science Publ., North Holland, 1991.
- [ZhaZZ92] Zhang, B., Zhang, L. and Zhang, H., "A Quantitative Analysis of the Behaviours of the PLN Network", *Neural Networks*, 5, pp. 639-44, 1992.
- [ZhaZZ93] Zhang, B., Zhang, L. and Zhang, H., "The Complexity of Learning in PLN Networks", Preprint, Dept. Computer Science, Tsinghua University, Beijing, China, 1993.

APPENDIX A

A.1. Proof of Lemma 4.1

The values $\psi_2(i)$ have been calculated (see Table 4.1):

$$\psi_2(0) = 2, \quad \psi_2(1) = 2, \quad \psi_2(2) = 10, \quad \psi_2(3) = 50, \quad \psi_2(4) = 250.$$

Therefore, expression (4.7) is correct for the value $D = 2$,

Equation (4.7) is now assumed to be correct for $D - 1$. This can be expressed by:

$$\psi_{D-1}(i) = \begin{cases} 2 & i = 0 \\ 2 \cdot 5^{i-1} & i = 1, 2, \dots, 2^{D-1}. \end{cases} \quad (\text{A.1})$$

Equation (4.7) is now derived, from (A.1), for any value D . The network can be thought of as made of a top layer 2-input RAM node, of which each input is connected to a regular pyramidal WANN of depth $D - 1$. The remaining of the proof is divided in two stages.

First the values $i = 0, 1, 2, \dots, 2^{D-1}$ are considered. Using the notations defined in Chapter 4, it holds that:

$$\psi_D(0) = \phi(0) = \psi_{D-1}(0)$$

and

$$\psi_D(i) = \phi(1) \cdot \psi'_{D-1}(i) = \psi_{D-1}(i),$$

for $i = 1, 2, \dots, 2^{D-1}$. Therefore (4.7) is verified for the values $i = 0, 1, 2, \dots, 2^{D-1}$,

Then, the values $i = 2^{D-1} + 1, \dots, 2^D$ are considered. It holds that:

$$\psi_D(i) = \phi(2) \cdot \psi'_{D-1}(2^{D-1}) \cdot \psi'_{D-1}(i - 2^{D-1}),$$

for $i = 2^{D-1} + 1, \dots, 2^D$. Let $j = i - 2^{D-1}$. Then, using (A.1):

$$\begin{aligned} \psi_D(j + 2^{D-1}) &= 10 \cdot \frac{2 \cdot 5^{2^{D-1}-1}}{2} \cdot \psi'_{D-1}(j) \\ &= 2 \cdot 5^{2^{D-1}} \cdot 5^{j-1} = 2 \cdot 5^{2^{D-1} + j - 1}, \end{aligned}$$

for $j = 1, 2, \dots, 2^{D-1}$. And finally:

$$\psi_p(i) = 2 \cdot 5^{i-1},$$

for $i = 2^{p-1} + 1, \dots, 2^p$, which shows that (4.7) is true for the values $i = 2^{p-1} + 1, \dots, 2^p$. This concludes the proof of (4.7).

A.2. Alternative proof of Theorem 4.1

Equation (4.8) can be verified by substitution into (4.1) and (4.2), with $N = 2$. From (4.1):

$$\Gamma(2, D) = \binom{2}{0} \cdot \phi(2) \cdot \left[\frac{\Gamma(2, D-1) - 2}{2} \right]^2 + \binom{2}{1} \cdot \phi(1) \cdot \left[\frac{\Gamma(2, D-1) - 2}{2} \right] + \binom{2}{2} \cdot \phi(0) \cdot 1.$$

From (4.2):

$$\phi(0) = 2, \quad \phi(1) = 2, \quad \phi(2) = 10.$$

Dropping the 2 in the notation $\Gamma(2, D)$, the following expression can be written:

$$\begin{aligned} \Gamma(D) &= 10 \cdot \left[\frac{\Gamma(D-1) - 2}{2} \right]^2 + 4 \cdot \left[\frac{\Gamma(D-1) - 2}{2} \right] + 2 \\ &= \frac{10}{4} [\Gamma^2(D-1) + 4 - 4\Gamma(D-1)] + 2\Gamma(D-1) - 4 + 2 \\ &= \frac{10}{4} \Gamma^2(D-1) - 8\Gamma(D-1) + 8. \end{aligned} \tag{A.2}$$

The substitution of $\Gamma(D-1)$ into (A.2), using (4.8), yields:

$$\Gamma(D) = \frac{10}{4} \cdot \left(\frac{8}{5} + \frac{2}{5} \cdot 6^{2^{D-1}} \right)^2 - 8 \cdot \left(\frac{8}{5} + \frac{2}{5} \cdot 6^{2^{D-1}} \right) + 8 = \frac{8}{5} + \frac{2}{5} \cdot 6^{2^D}$$

A.3. Proof of Lemma 4.2

If the terms corresponding to $j > 1$ are neglected, (4.2) becomes:

$$\phi(k) \approx 2^{2^k} - k \cdot \phi(k-1). \tag{A.3}$$

For example,

$$\begin{aligned}\phi(2) &\approx 2^{2^2} - 2 \cdot 2 = 12 \text{ (exact value} = 10\text{)}, \\ \phi(3) &\approx 2^{2^3} - 3 \cdot 12 = 220 \text{ (exact value} = 218\text{)}, \\ \phi(4) &\approx 2^{2^4} - 4 \cdot 220 = 64,656 \text{ (exact value} = 64,594\text{)}.\end{aligned}$$

Equation (A.3) can be expanded as

$$\phi(k) \approx \sum_{i=0}^k (-1)^i \frac{k!}{(k-i)!} 2^{2^{k-i}}$$

and if only the terms corresponding to $i = 0$ and $i = 1$ are retained:

$$\phi(k) \approx 2^{2^k} - k \cdot 2^{2^{k-1}}, \quad (\text{A.4})$$

For example, $\phi(2) \approx 8$, $\phi(3) \approx 208$, $\phi(4) \approx 64,512$. Finally, for $k \gg 1$, (A.4) can further be simplified to

$$\phi(k) \approx 2^{2^k}, \quad (\text{A.5})$$

A.4. Proof of Theorem 4.2

By only retaining, in (4.1), the term of the sum corresponding to $i = 0$, and by neglecting the (-2) term, (4.1) can be simplified to:

$$\Gamma(N, D) \approx \phi(N) \cdot \left[\frac{\Gamma(N, D-1)}{2} \right]^N, \quad (\text{A.6})$$

Using (A.5), it is useful to note that

$$\frac{\phi(N)}{2^N} \approx 2^{2^N - N} \approx 2^{2^N}, \quad (\text{A.7})$$

Utilising (A.7) and expanding (A.6) yields

$$\begin{aligned}\Gamma(N, D) &\approx 2^{2^N} \cdot 2^{2^N \cdot N} \cdot 2^{2^N \cdot N^2} \dots 2^{2^N \cdot N^{D-2}} \cdot \Gamma^{N^{D-1}}(N, 1) \\ &\approx 2^{2^N} \cdot 2^{2^N \cdot N} \cdot 2^{2^N \cdot N^2} \dots 2^{2^N \cdot N^{D-2}} \cdot 2^{2^N \cdot N^{D-1}},\end{aligned}$$

and therefore:

$$\log_2 \Gamma(N, D) \approx 2^N \sum_{i=0}^{D-1} N^i \quad (\text{A.8})$$

Finally, only the term of the sum corresponding to $i = D - 1$ is retained in (A.8):

$$\Gamma(N, D) \approx 2^{2^N N^{D-1}}.$$

This concludes the proof of Theorem 4.2.

APPENDIX B:

Proof of Theorem 4.3

Theorem 4.3 consists of an expression giving the number of ways, $F_{PCP}(N, D)$, of computing the parity checking function in a regular pyramidal WANN with topological parameters N and D . For simplicity, the number of ways of computing parity to 1 is calculated.

In a regular pyramidal network, trained to compute the parity function, each RAM performs either the 'parity to 1' function (outputting 1 if there is a odd number of 1's at its inputs) or the 'parity to 0' function (outputting 1 if there is a even number of 1's at its inputs). Hereafter, the functions 'parity to 1' and 'parity to 0' are referred to as ODD and EVEN, respectively. As an example, Figure B.1 shows the 4 solutions for computing the parity function in a 2-layer 2-input RAM pyramid, as given in [Ale89].

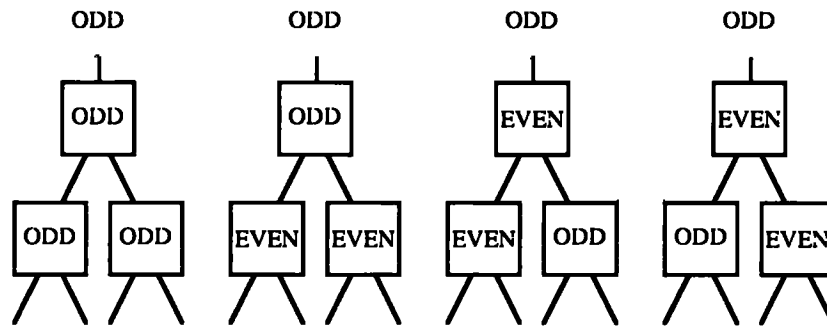


Fig. B.1: The 4 solutions for computing the parity function in a regular pyramidal WANN with parameters $N = D = 2$. Each pyramid RAM performs either the function EVEN or the function ODD.

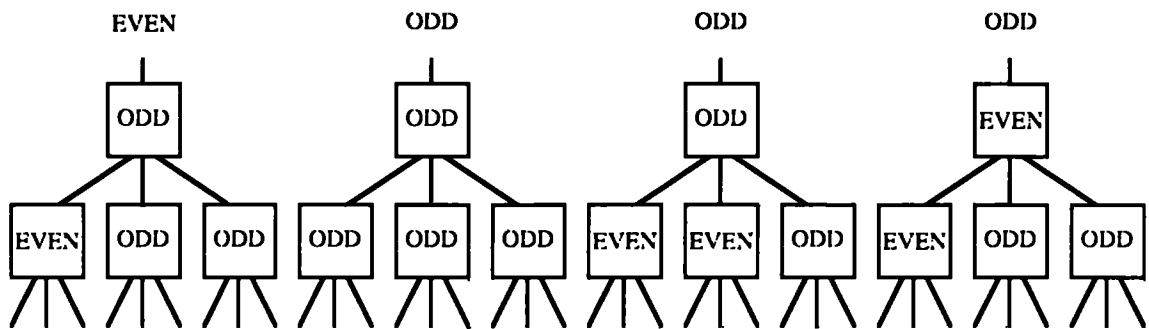


Fig. B.2: Four examples of the functions, either EVEN or ODD, performed by the RAMs of a regular pyramidal WANN with parameters $N = 3$ and $D = 2$. In this example, the first pyramid performs the function EVEN and the 3 others the function ODD.

In Figure B.2, four solution examples are given, for a pyramid with parameters $N = 3$ and $D = 2$, and performing either EVEN or ODD.

The following notations are used: $\mathbf{x}_i$ denotes the vector formed by the values outputted by the RAMs at layer i in the pyramid; ξ_i denotes the number of elements which have the value 1 in vector $\mathbf{x}_i$; ω_i denotes the number of RAMs performing ODD, in layer i of the pyramid. Figure B.3 shows a regular pyramidal WANN, with the indication of the different layers and vectors $\mathbf{x}_i$.

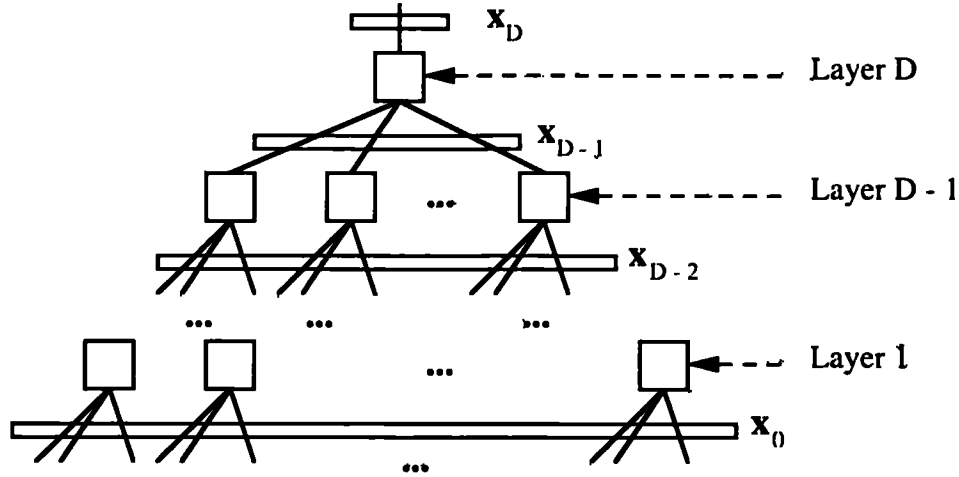


Fig. B.3: Graphical representation of a regular pyramidal WANN, where the different layers and the vectors $\mathbf{x}_i$ are indicated.

In the following derivation, use is made of a function $P(z_1, z_2, \dots, z_j, \dots)$ of a number of integer variables z_j , defined as

$$P(z_1, z_2, \dots, z_j, \dots) = \begin{cases} 1 & \text{if } \sum_j z_j \text{ is odd} \\ 0 & \text{otherwise.} \end{cases} \quad (\text{B.1})$$

Using (B.1), the relation between the number ξ_i of elements at 1 at the output of layer i , in a pyramid performing the ODD function, and the number ξ_{i-1} of elements at 1 at the output of layer $i-1$, can be expressed as:

$$P(\xi_i) = P(P(\xi_{i-1}), P(\omega_i), N) \quad (\text{B.2})$$

with

$$i = 1, \dots, D,$$

In addition to (B.2), 2 limit conditions can be stated,

$$P(\xi_D) = 1 \quad (\text{B.3})$$

and

$$P(\xi_0) = 1. \quad (\text{B.4})$$

which express the fact that the pyramid outputs the value 1 when a odd number of its inputs are at 1.

The following is a calculation of the number of arrangements of the variables which satisfy (B.2), (B.3) and (B.4). The calculation is carried out for N even, without loss of generality. In that case, (B.2) becomes:

$$P(\xi_i) = P(P(\xi_{i-1}), P(\omega_i)), \quad (\text{B.5})$$

with $i = 1, \dots, D$.

For $i = D$, there are 2 cases that satisfy (B.3) and (B.5):

- (a) $P(\omega_D) = 1$ and $P(\xi_{D-1}) = 0$;
- (b) $P(\omega_D) = 0$ and $P(\xi_{D-1}) = 1$.

Since layer D consists of a single RAM, there is therefore only one possible arrangement for case (a) (the RAM in layer D performing ODD) and also one possible arrangement for case (b) (the RAM in layer D performing EVEN).

For $i = D-1, \dots, 2$, depending on the level $i+1$, either $P(\xi_i) = 0$ is true or $P(\xi_i) = 1$ is true. The latter possibility is assumed, for simplicity. Again, 2 cases satisfy (B.5) and $P(\xi_i) = 1$:

- (a) $P(\omega_i) = 1$ and $P(\xi_{i-1}) = 0$;
- (b) $P(\omega_i) = 0$ and $P(\xi_{i-1}) = 1$.

Since layer i contains N^{D-i} RAMs, there are $2^{N^{D-i}-1}$ arrangements which satisfy (a) and an equal number which satisfies (b).

For $i = 1$, taking into account the condition (B.4), there is only one case to be considered, the one for which $P(\omega_1) = 0$ and $P(\xi_0) = 1$.

Finally, multiplying together the number of arrangements possible at each layer of the pyramid, the total number of ways of computing the parity function is derived. It holds

that:

$$\begin{aligned}
 F_p(N, D) &= \underbrace{2 \cdot 1}_D \cdot \underbrace{2 \cdot 2^{N-1}}_{D-1} \cdots \underbrace{2 \cdot 2^{N^{D-2}-1}}_4 \cdots \underbrace{2 \cdot 2^{N^{D-3}-1}}_2 \cdot \underbrace{1 \cdot 2^{N^{D-1}-1}}_1 \\
 &= 2^{\frac{N(N^{D-1}-1)}{D-1}}.
 \end{aligned}$$

which concludes the proof of Theorem 4.3.

APPENDIX C:**Calculation of the transition probability distributions (4.19)**

A regular pyramidal WANN, with parameters $N = D = 2$, is considered. The following notations are used: N_1 , N_2 and N_3 denote the top node, the bottom left node and the bottom right node, respectively. The function to learn is the parity function, denoted g . For a k -element binary input vector $\mathbf{i} = (i_{k-1}i_{k-2}\cdots i_1i_0)$, it holds that

$$g(\mathbf{i}) = \begin{cases} 1 & \text{if } \sum_j i_j \text{ is odd} \\ 0 & \text{if } \sum_j i_j \text{ is even.} \end{cases}$$

If, at each training step, the inputs to the pyramid are randomly chosen, then it holds that:

$$\Pr(g = 1) = \Pr(g = 0) = 0.5.$$

The variable U_i denotes the number of u values in node N_i . Correspondingly, the number of non- u values is denoted by $\overline{U}_i$. At training step t , it holds that

$$\overline{U}_i(t) = 2^N - U_i(t) = 4 - U_i(t).$$

The global variable U is defined as

$$U = \sum_i U_i.$$

It is useful to recall the initial conditions

$$U_i(0) = 4,$$

and therefore

$$U(0) = 12;$$

also the convergence condition

$$U(t) = 0, \quad \forall t > t_{conv}.$$

The following additional notations are used: $m_i(t)$ denotes the memory location addressed in node N_i during training step t ; $[m]$ denotes the contents of m ; finally, $\bar{u}$ denotes a non- u contents (either 1 or 0, which, in this case, are considered as a unique value).

It is assumed, for the simplicity of the analysis, that the probability v_i , that the addressed memory location in node N_i contains a u value, is proportional to the number U_i of u values in N_i :

$$v_i = \Pr([m_i] = u) = 2^{-N} \cdot U_i = \frac{U_i}{4}.$$

Similarly for the probability $\bar{v}_i$, that the addressed memory location in node N_i contains a $\bar{u}$ value, it holds that:

$$\bar{v}_i = \Pr([m_i] = \bar{u}) = 2^{-N} \cdot \bar{U}_i = \frac{\bar{U}_i}{4} = 1 - \frac{U_i}{4}.$$

In the top node N_1 , a $\bar{u}$ value is further differentiated between a value g , which is the parity value of the input pattern to the pyramid, and a value $\bar{g}$, which is the negation of g . It is assumed that the probabilities p and $\bar{p}$, of respectively addressing g or $\bar{g}$ in the top node, are proportional to $\bar{U}_1$:

$$p = \Pr([m] = g) = \alpha \cdot \bar{v}_1 = \alpha \cdot \frac{\bar{U}_1}{4}$$

and

$$\bar{p} = \Pr([m] = \bar{g}) = \beta \cdot \bar{v}_1 = \beta \cdot \frac{\bar{U}_1}{4},$$

with

$$\alpha + \beta = 1.$$

Furthermore, it is assumed that, the more $\bar{u}$ values in the top node, the more likely they are to be g values. This can be expressed as:

$$\alpha = \frac{1}{2} + \frac{\bar{v}_1}{2}$$

and

$$\beta = \frac{1}{2} - \frac{\bar{v}_1}{2}.$$

And finally:

$$p = \frac{1}{2}(1 + \bar{v}_1)\bar{v}_1$$

and

$$\bar{p} = \frac{1}{2}(1 - \bar{v}_1)\bar{v}_1,$$

It is useful to note the relation:

$$p + \bar{p} + v_1 = 1,$$

In the following, all the cases leading to different ΔU_i transitions are considered, together with their probability of occurrence. For the bottom layer, 4 cases are to be considered, corresponding to whether the memory location addressed in each of the bottom nodes contains u or $\bar{u}$. Each of these cases leads to a number t of locations addressable in the top node. The contents of these locations could be g (with probability p), $\bar{g}$ (also with probability p) or u (with probability v_1). Thus, if a case in the bottom layer yields t addressable locations in the top node, then $\binom{t+2}{2}$ cases need to be considered in the top node, each of which has a probability given by the trinomial distribution [Gri82]:

$$\frac{t!}{n_g! \cdot n_{\bar{g}}! \cdot n_u!} \cdot p^{n_g} \cdot p^{n_{\bar{g}}} \cdot v_1^{n_u},$$

with

$$t = n_g + n_{\bar{g}} + n_u$$

and in which n_g , $n_{\bar{g}}$ and n_u are the number of addressable locations in the top node which contain the values g , $\bar{g}$ and u , respectively.

For the bottom layer, the following table is obtained:

	$[m_2]$	$[m_3]$	t	Probability
case 1	$\bar{u}$	$\bar{u}$	1	$\bar{v}_2 \cdot \bar{v}_3$
case 2	$\bar{u}$	u	2	$\bar{v}_2 \cdot v_3$
case 3	u	$\bar{u}$	2	$v_2 \cdot \bar{v}_3$
case 4	u	u	4	$v_2 \cdot v_3$

Table C.1: The 4 cases to be considered in the bottom layer of the pyramid, corresponding to whether the memory location addressed in each of the bottom nodes contains u or $\bar{u}$.

The first case of the bottom layer leads to the following cases in the top node:

	n_{κ}	$n_{\bar{\kappa}}$	n_u	Probability	ΔU_1	ΔU_2	ΔU_3	ΔU
case 1.1	1	0	0	p	0	0	0	0
case 1.2	0	1	0	$\bar{p}$	+1	+1	+1	+3
case 1.3	0	0	1	v_1	-1	0	0	-1

Table C.2: The different cases to be considered in the top node when the memory location addressed in each of the bottom nodes contains $\bar{u}$.

The second case of the bottom layer leads to the following cases in the top node:

	n_{κ}	$n_{\bar{\kappa}}$	n_u	Probability	ΔU_1	ΔU_2	ΔU_3	ΔU
case 2.1	2	0	0	p^2	0	-1	0	-1
case 2.2	1	1	0	$2p \cdot \bar{p}$	0	-1	0	-1
case 2.3.1	1	0	1	$2p \cdot v_1 \cdot \frac{2}{3}$	0	-1	0	-1
case 2.3.2				$2p \cdot v_1 \cdot \frac{1}{3}$	-1	-1	0	-2
case 2.4	0	2	0	$\bar{p}^2$	+1	0	+1	+2
case 2.5	0	1	1	$2\bar{p} \cdot v_1$	-1	-1	0	-2
case 2.6	0	0	2	v_1^2	-1	-1	0	-2

Table C.3: The different cases to be considered in the top node when the memory location addressed in the bottom left node contains $\bar{u}$ and the memory location addressed in the bottom right node contains u .

The case 2.3 is made of 2 different sub-cases. The case 2.3.1 corresponds to a training step

during which only the addressed u value in the bottom right node is set to a $\bar{u}$ value. The case 2.3.2 corresponds to a training step during which the addressed u value in the bottom right node is set to a $\bar{u}$ value and the addressable u value in the top node is set to a g value.

The third case of the bottom layer is similar to the second case and leads to the following cases in the top node:

	n_r	$n_{\bar{r}}$	n_u	Probability	ΔU_1	ΔU_2	ΔU_3	ΔU
case 3.1	2	0	0	p^2	0	0	-1	-1
case 3.2	1	1	0	$2p \cdot \bar{p}$	0	0	-1	-1
case 3.3.1	1	0	1	$2p \cdot v_1 \cdot \frac{2}{3}$	0	0	-1	-1
case 3.3.2				$2p \cdot v_1 \cdot \frac{1}{3}$	-1	0	-1	-2
case 3.4	0	2	0	$\bar{p}^2$	+1	+1	0	+2
case 3.5	0	1	1	$2\bar{p} \cdot v_1$	-1	0	-1	-2
case 3.6	0	0	2	v_1^2	-1	0	-1	-2

Table C.4: The different cases to be considered in the top node when the memory location addressed in the bottom left node contains u and the memory location addressed in the bottom right node contains $\bar{u}$.

The fourth case of the bottom layer leads to the following cases in the top node:

	n_k	$n_{\bar{k}}$	n_u	Probability	ΔU_1	ΔU_2	ΔU_3	ΔU
case 4.1	4	0	0	p^4	0	-1	-1	-2
case 4.2	3	1	0	$4p^3 \cdot \bar{p}$	0	-1	-1	-2
case 4.3.1	3	0	1	$4p^3 \cdot v_1 \cdot \frac{6}{7}$	0	-1	-1	-2
case 4.3.2				$4p^3 \cdot v_1 \cdot \frac{1}{7}$	-1	-1	-1	-3
case 4.4	2	2	0	$6p^2 \cdot \bar{p}^2$	0	-1	-1	-2
case 4.5.1	2	1	1	$12p^2 \cdot \bar{p} \cdot v_1 \cdot \frac{4}{5}$	0	-1	-1	-2
case 4.5.2				$12p^2 \cdot \bar{p} \cdot v_1 \cdot \frac{1}{5}$	-1	-1	-1	-3
case 4.6.1	2	0	2	$6p^2 \cdot v_1^2 \cdot \frac{4}{6}$	0	-1	-1	-2
case 4.6.2				$6p^2 \cdot v_1^2 \cdot \frac{2}{6}$	-1	-1	-1	-3
case 4.7	1	3	0	$4p \cdot \bar{p}^3$	0	-1	-1	-2
case 4.8.1	1	2	1	$12p \cdot \bar{p}^2 \cdot v_1 \cdot \frac{2}{3}$	0	-1	-1	-2
case 4.8.2				$12p \cdot \bar{p}^2 \cdot v_1 \cdot \frac{1}{3}$	-1	-1	-1	-3
case 4.9.1	1	1	2	$12p \cdot \bar{p} \cdot v_1^2 \cdot \frac{1}{2}$	0	-1	-1	-2
case 4.9.2				$12p \cdot \bar{p} \cdot v_1^2 \cdot \frac{1}{2}$	-1	-1	-1	-3
case 4.10.1	1	0	3	$4p \cdot v_1^3 \cdot \frac{2}{5}$	0	-1	-1	-2
case 4.10.2				$4p \cdot v_1^3 \cdot \frac{3}{5}$	-1	-1	-1	-3
case 4.11	0	4	0	$\bar{p}^4$	+1	0	0	+1
case 4.12	0	3	1	$4\bar{p}^3 \cdot v_1$	-1	-1	-1	-3
case 4.13	0	2	2	$6\bar{p}^2 \cdot v_1^2$	-1	-1	-1	-3
case 4.14	0	1	3	$4\bar{p} \cdot v_1^3$	-1	-1	-1	-3
case 4.15	0	0	4	v_1^4	-1	-1	-1	-3

Table C.5: The different cases to be considered in the top node when the memory location addressed in each of the bottom nodes contains u .

In order to calculate the distribution probabilities $p_x(\xi)$, all the cases corresponding to the possible values of U_1 , U_2 and U_3 are considered. For each case, the probabilities of the

different values of ΔU are calculated using the results in the Tables C.1 to C.5. The resulting probabilities are then averaged over the different cases corresponding to a same value of U_* .

As an example, the calculation of the distribution $p_{11}(\xi)$ is detailed. Writing the values of U_1, U_2, U_3 as triplets (U_1, U_2, U_3) , 3 cases need to be considered: (3,4,4), (4,3,4) and (4,4,3). For each of these 3 triplets, the probabilities $p_{11}(\xi)$ are calculated using the previous formulae and tables.

For (3,4,4), the following distribution is obtained:

$$\begin{aligned} p_{11}(-3) &= 4p^3 \cdot v_1 \cdot \frac{1}{7} + 12p^2 \cdot \bar{p} \cdot v_1 \cdot \frac{1}{5} + 6p^2 \cdot v_1^2 \cdot \frac{2}{6} \\ &\quad + 12p \cdot \bar{p}^2 \cdot v_1 \cdot \frac{1}{3} + 12p \cdot \bar{p} \cdot v_1^2 \cdot \frac{1}{2} + 4p \cdot v_1^3 \cdot \frac{3}{5} \\ &\quad + 4\bar{p}^3 \cdot v_1 + 6\bar{p}^2 \cdot v_1^2 + 4\bar{p} \cdot v_1^3 + v_1^4 \\ &= 0.75172; \end{aligned}$$

$$p_{11}(-2) = 0.24820;$$

$$p_{11}(1) = 0.00008;$$

$$p_{11}(-1) = p_{11}(0) = p_{11}(2) = p_{11}(3) = 0.$$

Similarly, for both (4,3,4) and (4,4,3):

$$p_{11}(-3) = 0.75;$$

$$p_{11}(-2) = 0.25;$$

$$p_{11}(-1) = p_{11}(0) = p_{11}(1) = p_{11}(2) = p_{11}(3) = 0.$$

Making the assumption that the 3 triplets are equally likely to occur, the final transition probability distribution is obtained:

$$p_{11}(-3) = \frac{1}{3}(0.7517 + 2 \cdot 0.75) = 0.75058;$$

$$p_{11}(-2) = \frac{1}{3}(0.2482 + 2 \cdot 0.25) = 0.24940;$$

$$p_{11}(1) = \frac{1}{3}(0.00008 + 2 \cdot 0.0) = 0.00003;$$

$$p_{11}(-1) = p_{11}(0) = p_{11}(2) = p_{11}(3) = 0.$$

The remaining probability distributions are calculated using a computer. The number of triplets to be considered, for each U , is denoted by $\tau_3(U)$. These values are given by the coefficients of the generating function [PolTW83]:

$$(1 + x + x^2 + x^3 + x^4)^3,$$

such that

$$\sum_{U=0}^{12} \tau_3(U) \cdot x^U = (1 + x + x^2 + x^3 + x^4)^3 = \frac{(1 - x^5)^3}{(1 - x)^3}.$$

The resulting $\tau_3(U)$ values are shown in Table C.6.

U	0	1	2	3	4	5	6	7	8	9	10	11	12
$\tau_3(U)$	1	3	6	10	15	18	19	18	15	10	6	3	1

Table C.6: The number $\tau_3(U)$ of triplets to be considered, for each value of U .

In the general case of a pyramidal WANN, with parameters N and D , the corresponding numbers $\tau_{N,D}(U)$ would be given by:

$$\sum_{U=0}^{2^N \cdot \frac{N^D-1}{N-1}} \tau_{N,D}(U) \cdot x^U = \left(\frac{1 - x^{2^N+1}}{1 - x} \right)^{\frac{N^D-1}{N-1}}.$$

For the case $N = D = 2$, the resulting transition probability matrix is given by (4.19). The corresponding probability distributions are given in Table C.7 and are also represented graphically in Figure 4.3.

U	$\xi = -3$	$\xi = -2$	$\xi = -1$	$\xi = 0$	$\xi = 1$	$\xi = 2$	$\xi = 3$
0	0	0	0	1	0	0	0
1	0	0	.25	.71875	0	0	.03125
2	0	.02865	.43937	.48698	0	.00073	.04427
3	.00116	.08517	.5695	.29941	0	.00199	.04277
4	.00499	.16585	.64473	.15104	0	.00319	.03021
5	.01536	.30864	.57497	.07628	.00001	.00468	.02008
6	.04032	.45967	.44856	.03454	.00001	.0054	.01151
7	.0876	.58837	.3011	.01291	.00003	.00467	.00532
8	.17011	.65902	.16139	.00417	.00004	.00319	.00208
9	.32904	.58458	.08278	.00098	.00006	.00199	.00059
10	.52391	.44739	.02791	0	0	.00073	0
11	.75058	.2494	0	0	.00003	0	0
12	1.0	0	0	0	0	0	0

Table C.7: Transition probability distributions.

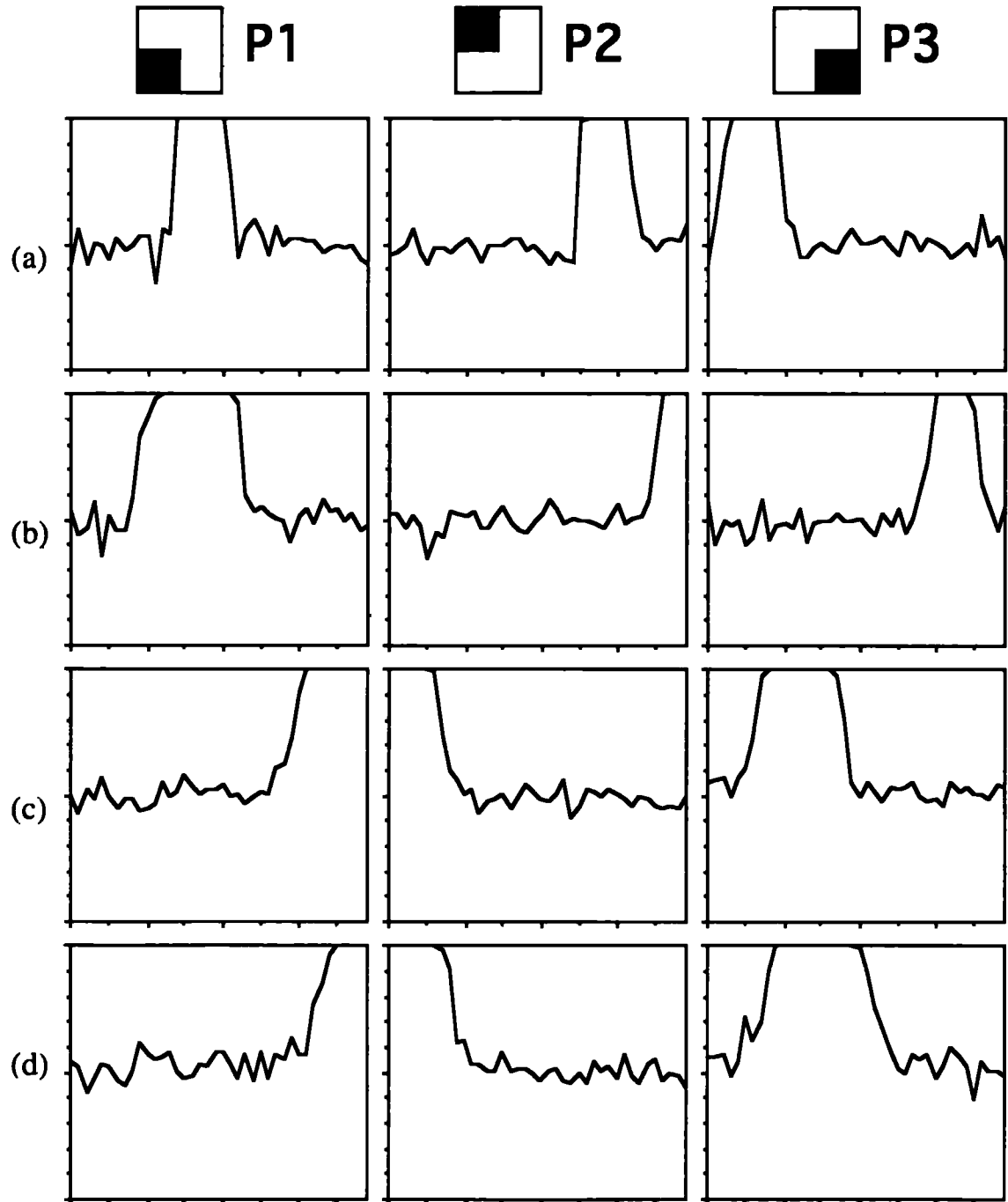
APPENDIX D:**Parameter variations in the C-discriminator network****D.1. Excitation radius**

Fig. D.1: Response maps produced by the presentation of patterns P1, P2 and P3, after 1000 training steps, with noise level = 4 and excitation radiuses (a) 2, (b) 5, (c) 10, (d) 20.

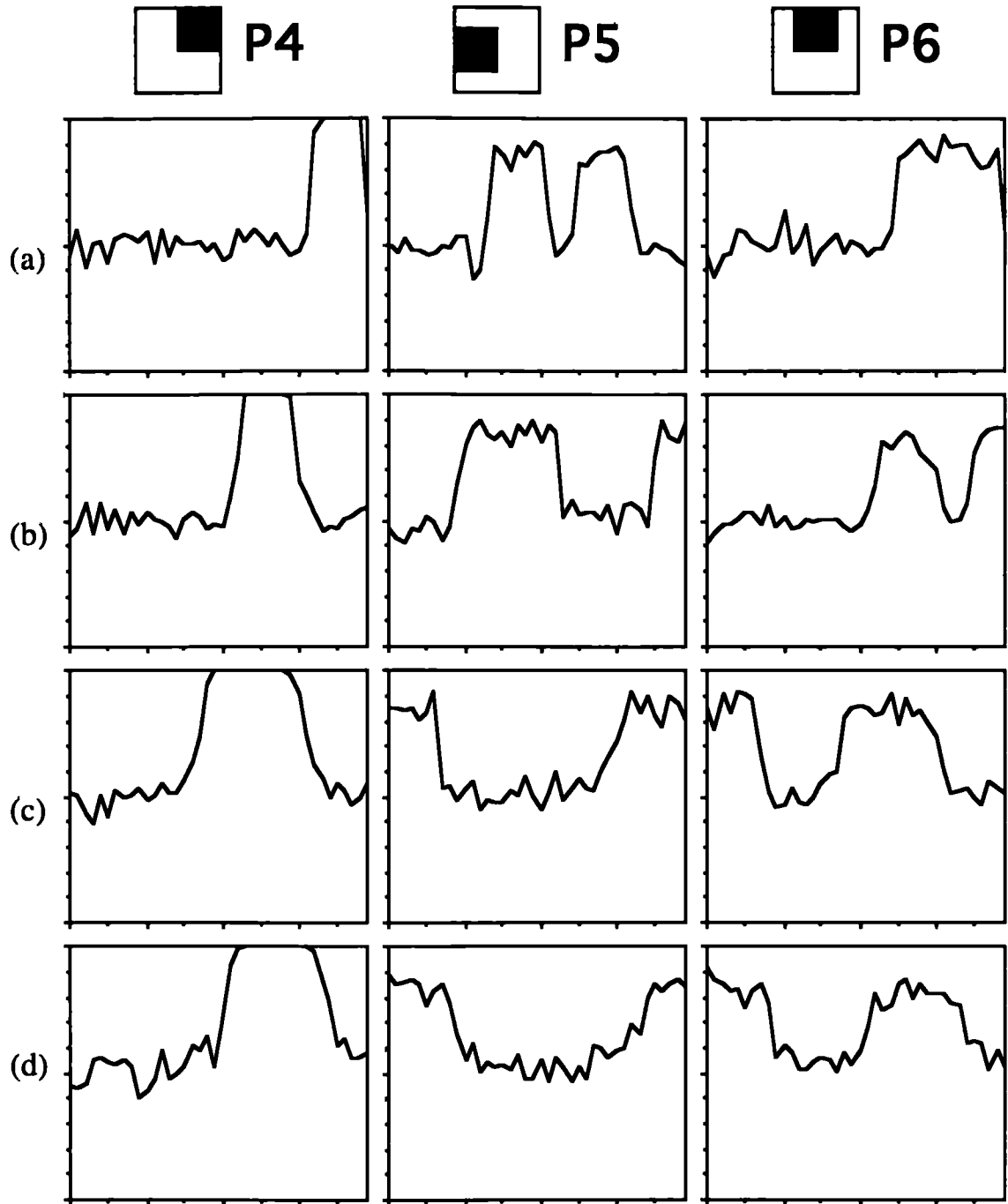


Fig. D.2: Response maps produced by the presentation of patterns P4, P5 and P6, after 1000 training steps, with noise level = 4 and excitation radiuses (a) 2, (b) 5, (c) 10, (d) 20.

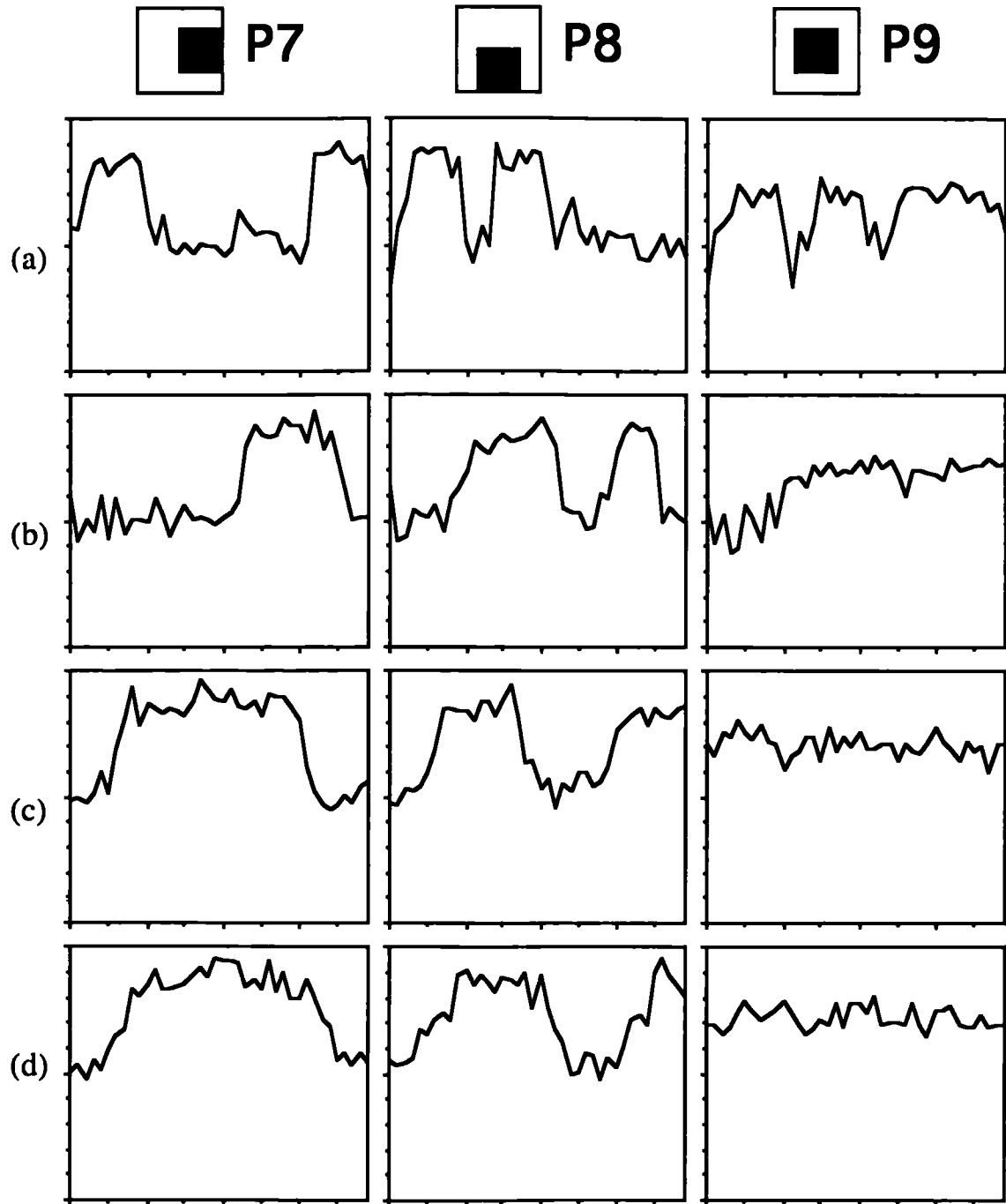


Fig. D.3: Response maps produced by the presentation of patterns P7, P8 and P9, after 1000 training steps, with noise level = 4 and excitation radiuses (a) 2, (b) 5, (c) 10, (d) 20.

D.2. Noise level

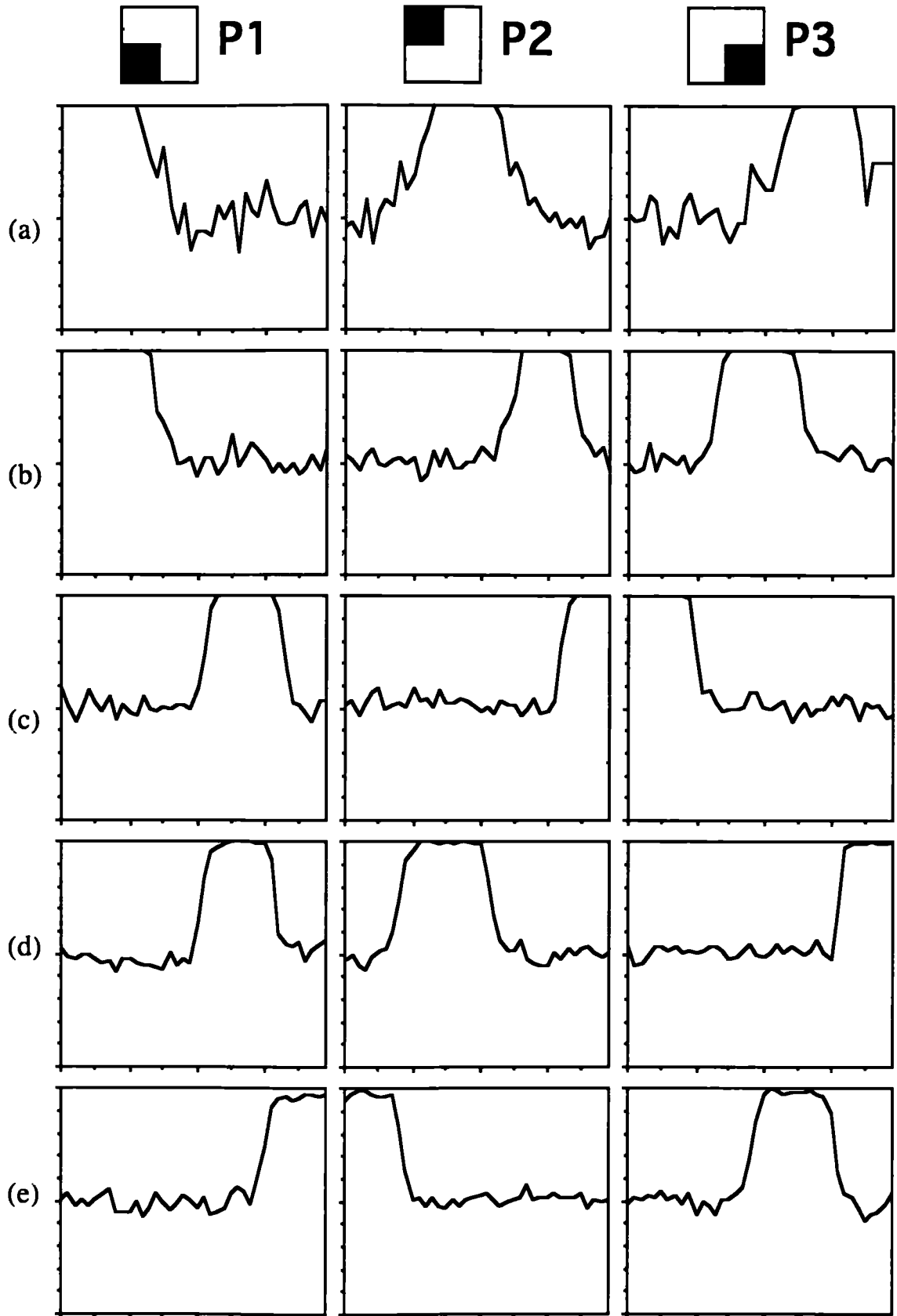


Fig. D.4: Response maps produced by patterns P1, P2 and P3, after 1000 training steps, with excitation radius = 10 and noise levels (a) 0, (b) 4, (c) 10, (d) 20 and (e) 32.

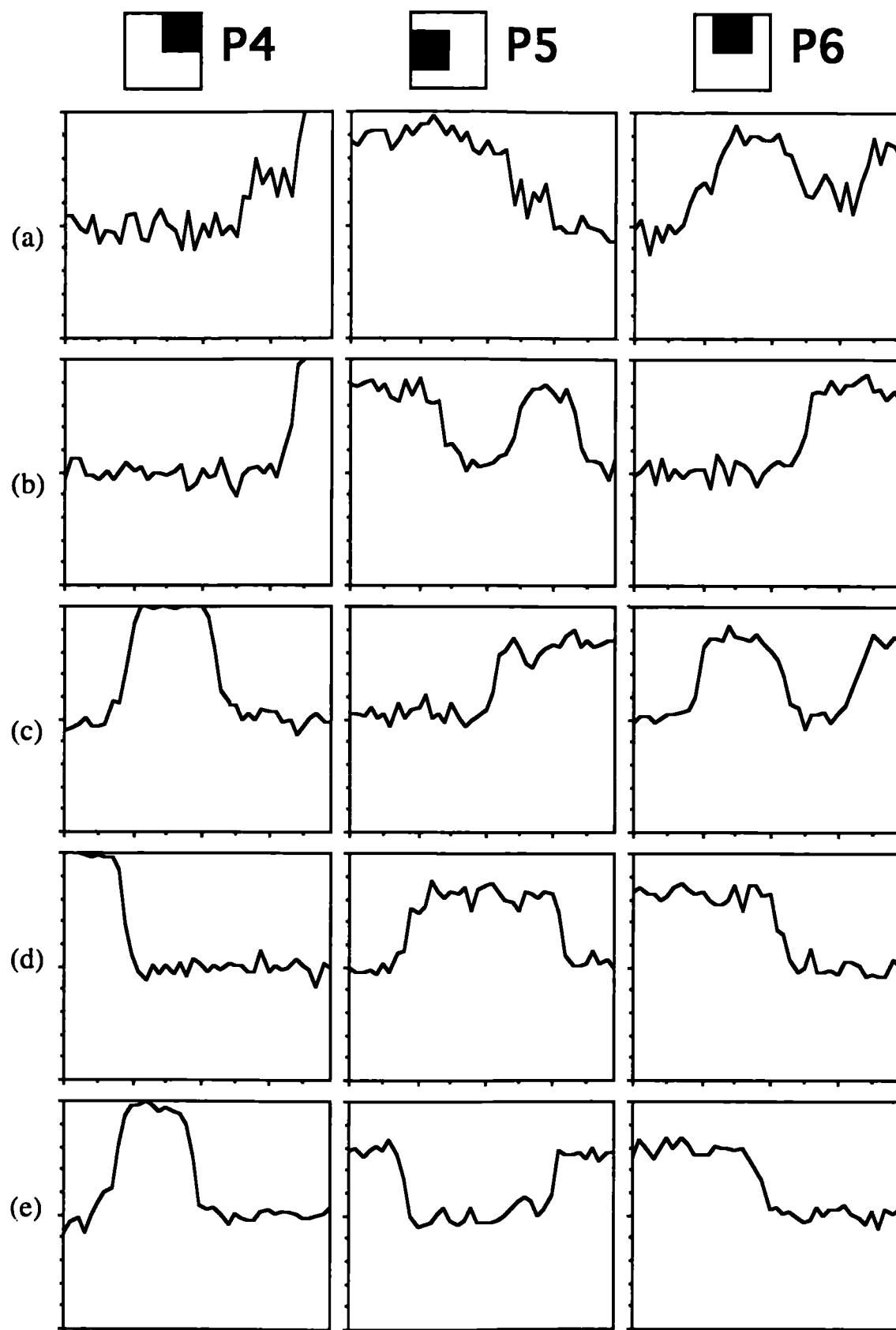


Fig. D.5: Response maps produced by patterns P4, P5 and P6, after 1000 training steps, with excitation radius = 10 and noise levels (a) 0, (b) 4, (c) 10, (d) 20 and (e) 32.

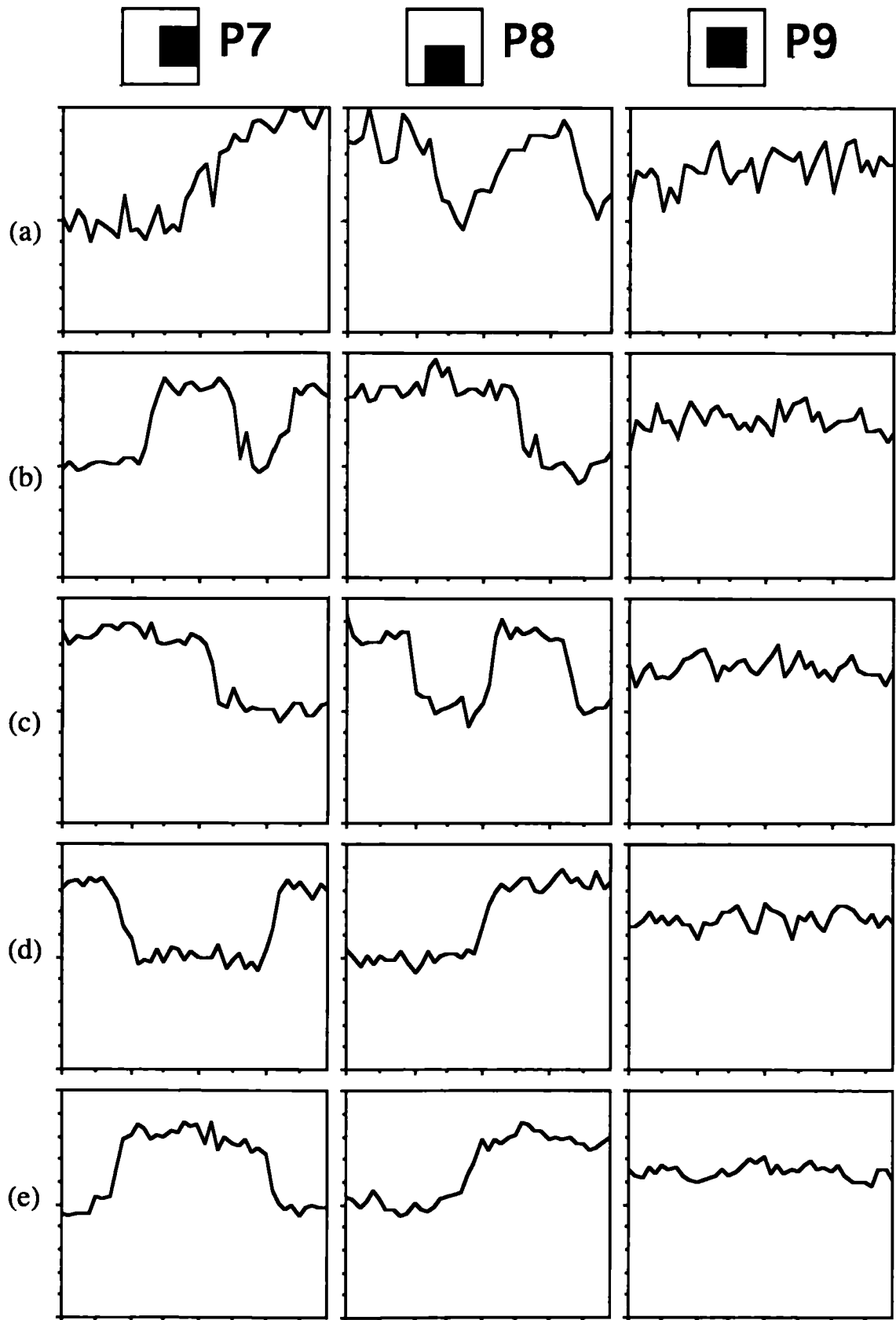


Fig. D.6: Response maps produced by patterns P7, P8 and P9, after 1000 training steps, with excitation radius = 10 and noise levels (a) 0, (b) 4, (c) 10, (d) 20 and (e) 32.

APPENDIX E:**Proof of Theorem 7.4**

In this appendix, a proof of (7.15) is given, stating the equivalence, in the case of two opposite stored patterns, of the retrieval equation for an auto-associative GNU with $(2n - 1)$ -input nodes to that for a GNU with $2n$ -input nodes. This can be stated as:

$$f(x(t), 2n - 1) = f(x(t), 2n), \quad (\text{E.1})$$

in which, dropping the dependency in t and using (7.14),

$$f(x, 2n - 1) = \sum_{j=n}^{2n-1} \binom{2n-1}{j} \cdot x^j (1-x)^{2n-1-j} = A(n) \quad (\text{E.2})$$

and

$$f(x, 2n) = \sum_{j=n+1}^{2n} \binom{2n}{j} \cdot x^j (1-x)^{2n-j} + \frac{1}{2} \cdot \binom{2n}{n} \cdot x^n (1-x)^n = B(n), \quad (\text{E.3})$$

First, (E.1) is shown to be true for $n = 1$, In that case

$$A(1) = \sum_{j=1}^1 \binom{1}{j} \cdot x^j (1-x)^{1-j} = x,$$

and

$$B(1) = \sum_{j=2}^2 \binom{2}{j} \cdot x^j (1-x)^{2-j} + \frac{1}{2} \cdot \binom{2}{1} \cdot x(1-x) = x^2 + x - x^2 = x.$$

Then, it is shown that, if

$$A(n) = B(n),$$

then

$$A(n+1) = B(n+1).$$

Using (E.2), an expression for $A(n+1)$ can be written as:

$$A(n+1) = \sum_{j=n+1}^{2(n+1)-1} \binom{2(n+1)-1}{j} \cdot x^j (1-x)^{2(n+1)-j-1}. \quad (\text{E.4})$$

Using twice the formula

$$\binom{m}{r} = \binom{m-1}{r} + \binom{m-1}{r-1}, \quad (\text{E.5})$$

the combinatorial coefficient in (E.4) can be written as:

$$\begin{aligned} \binom{2(n+1)-1}{j} &= \binom{2n}{j} + \binom{2n}{j-1} \\ &= \binom{2n-1}{j} + \binom{2n-1}{j-1} + \binom{2n-1}{j-1} + \binom{2n-1}{j-2} \\ &= \binom{2n-1}{j} + 2 \cdot \binom{2n-1}{j-1} + \binom{2n-1}{j-2}. \end{aligned} \quad (\text{E.6})$$

The substitution of (E.6) into (E.4) yields

$$\begin{aligned} A(n+1) &= \sum_{j=n+1}^{2(n+1)-1} \binom{2n-1}{j} \cdot x^j (1-x)^{2(n+1)-j-1} \\ &\quad + 2 \cdot \sum_{j=n+1}^{2(n+1)-1} \binom{2n-1}{j-1} \cdot x^j (1-x)^{2(n+1)-j-1} \\ &\quad + \sum_{j=n+1}^{2(n+1)-1} \binom{2n-1}{j-2} \cdot x^j (1-x)^{2(n+1)-j-1} \\ &= (1-x)^2 \cdot \sum_{j=n+1}^{2(n+1)-1} \binom{2n-1}{j} \cdot x^j (1-x)^{2n-1-j} \\ &\quad + 2x(1-x) \cdot \sum_{l=n}^{2n} \binom{2n-1}{l} \cdot x^l (1-x)^{2n-1-l} \\ &\quad + x^2 \cdot \sum_{k=n-1}^{2n-1} \binom{2n-1}{k} \cdot x^k (1-x)^{2n-1-k} \\ &= (1-x)^2 \cdot C(n) - (1-x)^2 \binom{2n-1}{n} \cdot x^n (1-x)^{n-1} \\ &\quad + 2x(1-x) \cdot C(n) \end{aligned}$$

$$+ x^2 \cdot C(n) + x^2 \cdot \binom{2n-1}{n-1} \cdot x^{n-1} (1-x)^n,$$

with

$$\begin{aligned} C(n) &= \sum_{l=n}^{2n} \binom{2n-1}{l} \cdot x^l (1-x)^{2n-1-l} \\ &= \sum_{l=n}^{2n-1} \binom{2n-1}{l} \cdot x^l (1-x)^{2n-1-l} = A(n). \end{aligned}$$

This yields

$$\begin{aligned} A(n+1) &= A(n) - \binom{2n-1}{n} \cdot x^n (1-x)^{n+1} + \binom{2n-1}{n-1} \cdot x^{n+1} (1-x)^n \\ &= A(n) - \binom{2n-1}{n} \cdot x^n (1-x)^{n+1} + \binom{2n-1}{n} \cdot x^{n+1} (1-x)^n \\ &= A(n) - \binom{2n-1}{n} \cdot x^n (1-x)^n \cdot (1-2x). \end{aligned} \quad (\text{E.7})$$

Similarly, using (E.3), an expression is developed for $B(n+1)$:

$$\begin{aligned} B(n+1) &= \sum_{j=(n+1)+1}^{2(n+1)} \binom{2(n+1)}{j} \cdot x^j (1-x)^{2(n+1)-j} \\ &\quad + \frac{1}{2} \binom{2(n+1)}{n+1} \cdot x^{n+1} (1-x)^{n+1} \\ &= D(n) + \frac{1}{2} \binom{2(n+1)}{n+1} \cdot x^{n+1} (1-x)^{n+1}. \end{aligned} \quad (\text{E.8})$$

$D(n)$ is then developed in a similar way to that done above for $A(n+1)$:

$$\begin{aligned} D(n) &= \sum_{j=n+2}^{2n+2} \binom{2n}{j} \cdot x^j (1-x)^{2n+2-j} + 2 \sum_{j=n+2}^{2n+2} \binom{2n}{j-1} \cdot x^j (1-x)^{2n+2-j} \\ &\quad + \sum_{j=n+2}^{2n+2} \binom{2n}{j-2} \cdot x^j (1-x)^{2n+2-j} \end{aligned}$$

$$\begin{aligned}
&= (1-x)^2 \sum_{j=n+2}^{2n+2} \binom{2n}{j} \cdot x^j (1-x)^{2n-j} + 2x(1-x) \sum_{l=n+1}^{2n+1} \binom{2n}{l} \cdot x^l (1-x)^{2n-l} \\
&\quad + x^2 \sum_{k=n}^{2n} \binom{2n}{k} \cdot x^k (1-x)^{2n-k} \\
&= E(n) - \binom{2n}{n+1} \cdot x^{n+1} (1-x)^{n+1} + \binom{2n}{n} \cdot x^{n+2} (1-x)^n, \tag{E.9}
\end{aligned}$$

with

$$E(n) = B(n) - \frac{1}{2} \binom{2n}{n} \cdot x^n (1-x)^n$$

and because of the hypothesis

$$A(n) = B(n),$$

$$E(n) = A(n) - \frac{1}{2} \binom{2n}{n} \cdot x^n (1-x)^n. \tag{E.10}$$

Then, taking into account (E.9) and (E.10), (E.8) becomes:

$$\begin{aligned}
B(n+1) &= A(n) - \frac{1}{2} \binom{2n}{n} \cdot x^n (1-x)^n - \binom{2n}{n+1} \cdot x^{n+1} (1-x)^{n+1} \\
&\quad + \binom{2n}{n} \cdot x^{n+2} (1-x)^n + \frac{1}{2} \binom{2n+2}{n+1} \cdot x^{n+1} (1-x)^{n+1}. \tag{E.11}
\end{aligned}$$

Grouping, in the right member of (E.8), the first term with the third term, and the second term with the fourth term, yields

$$\begin{aligned}
B(n+1) &= A(n) + \binom{2n}{n} \cdot x^n (1-x)^n \left(-\frac{1}{2} + x^2 \right) \\
&\quad + x^{n+1} (1-x)^{n+1} \cdot \left[-\binom{2n}{n+1} + \frac{1}{2} \binom{2n}{n+1} + \binom{2n}{n} + \frac{1}{2} \binom{2n}{n-1} \right]
\end{aligned}$$

$$\begin{aligned}
&= A(n) - \frac{1}{2} \binom{2n}{n} \cdot x^n \cdot (1-x)^n \cdot (1-2x) \\
&= A(n) - \frac{1}{2} \left[\binom{2n-1}{n} + \binom{2n-1}{n-1} \right] \cdot x^n \cdot (1-x)^n \cdot (1-2x) \\
&= A(n) - \binom{2n-1}{n} \cdot x^n (1-x)^n \cdot (1-2x),
\end{aligned}$$

And finally, taking into account (E.7), it holds that

$$B(n+1) = A(n+1),$$

which concludes the proof of Theorem 7.4.

APPENDIX F:**Published papers****F.1. Proceedings IJCNN-90, San Diego, June 1990:**

“Self-organisation properties of a discriminator-based neural network”.

F.2. Proceedings IJCNN-91, Seattle, July 1991:

“Storage capacity and retrieval properties of an auto-associative general neural unit”.

F.3. Proceedings ICANN-92, Brighton, September 1992:

“Parallel hardware implementation of the general neural unit model”.

F.4. Proceedings ICANN-93, Amsterdam, September 1993:

“A dynamically generalising weightless neural element”.

SELF-ORGANIZATION PROPERTIES OF A DISCRIMINATOR-BASED NEURAL NETWORK

Panayotis Ntounioufis

Neural Systems Engineering Laboratory
Imperial College, London SW7 2BT, England
email: JANET panos%winge@sig.ee.ic.ac.uk

ABSTRACT

Organized maps coding sensory information have been found in the brain. Kohonen showed that it is possible to model a self-organizing system capable of forming a reduced representation of input information into such maps. The aim of this paper is to define a network with identical properties but based on a RAM neuron model, namely the C-discriminator, whereas Kohonen uses the classical linear neuron model. Such RAM models have been studied before but only within the context of a supervised training scheme whereas this paper presents an unsupervised training algorithm. The architecture of the network is presented and the training procedure is defined. It is shown, on a simple example, that the net behaviour can be predicted considering the overlap areas between input patterns.

1. INTRODUCTION

Physiological studies have shown that sensory information is encoded by the brain, at least in the primary sensory areas, in various geometrically organized maps [2]. Sensory experiences such as topographic coordinates of the body or pitch of audible tones are mapped into a two-dimensional coordinate system defined on some areas of the cortex. These areas are composed of two-dimensional layers of neurons in which there is dense lateral interconnection with relatively few interlayer connections [3]. The degree of lateral interaction between cells is usually described as having the form of a Mexican hat. Three types of lateral interaction can be distinguished: a short range lateral *excitation* surrounded by a penumbral region of *inhibitory action* which is surrounded itself by a weaker *excitatory action* [3]. Kohonen discovered that a *self-organizing* array of neural cells with modifiable parameters is capable of forming a *topologically ordered map* of data to which it has been exposed, automatically forming a reduced representation of input information, involving feature extraction. This paradigm was successfully applied to speech recognition by Kohonen [2][3].

In this paper, we demonstrate a neural network showing similar self-organizing properties to a Kohonen net but using a particular model of neuron, the *discriminator* model, whereas Kohonen uses a conventional linear neuron model. The latter is a variation of the Mc Culloch and Pitts neuron. It is essentially an m -input device with input values labelled $i_1, i_2, \dots, i_m$, where such values are real numbers. Each input i_j is multiplied by some *synaptic weight* w_j and the entire neuron reacts by producing an output response S equal to the scalar product of the input vector i and the weight vector w of the neuron, S being considered as a measure of similarity between those two vectors.

The *discriminator* was first introduced by Aleksander and used in the WISARD recognition system, the first industrial neural computer [1]. This model is based on the *N-tuple sampling* paradigm of Bledsoe and Browning [4]. Conceptually a discriminator is composed of an array of RAM (Random Access Memory) elements. Each of those RAMs is addressed by a *n-tuple* which is a number n of bits taken from an array of m binary values forming the input to the discriminator. The adaptive parameters of a discriminator are the values stored in the memory locations of its RAMs. They are updated through a training procedure. The discriminator output or *response* p is obtained by summing the values held in the storage locations addressed by the n -tuples.

There are two main differences between the discriminator model used in our work,

- 2 -

hereafter referred to as *C-discriminator* (continuously valued discriminator), and the one used in Wisard. Firstly, the storage locations in Wisard discriminators hold binary values (1's and 0's) whereas in C-discriminators they are allowed to store a continuous range of values between 0 and 1. Secondly, in Wisard each discriminator is assigned to a certain class of input patterns whereas in our system each input pattern is seen by all C-discriminators during the training of the system. In fact, the class to which the presented patterns belong is not known *a priori* and, in that sense, we can consider the training as *unsupervised* as opposed to a *supervised* training in Wisard.

We describe and analyse our model, and show, on a simple example, that it is possible to use a C-discriminator network to form a topologically ordered map of input data, resulting in a *clustering* of responses to similar patterns in specific regions of the output map. It is argued that it is possible to predict the net responses to different input patterns considering the overlap areas between them. Finally, we show the temporal evolution of responses to a certain pattern and the formation of specific areas of activity in the output map.

2. THE DISCRIMINATOR-BASED NETWORK

2.1. The network

The overall structure of the network considered here is similar to that of a Kohonen net. The net is made of one layer of processing units or neurons which are modelled as C-discriminators. The units are labelled

$$C_1, C_2, \dots, C_\gamma, \dots, C_Q$$

and are arranged as a one or two-dimensional array. An example of the one-dimensional arrangement is shown in Figure 1.

An important characteristic of this structure is that each C-discriminator sees the whole of any input pattern presented to the net. The input patterns consist of arrays of m black or white pixels which are respectively translated into 1's and 0's.

Each C-discriminator produces an output response ρ and the set of responses yields an output map. During training of the network, the neurons are interacting with each other through lateral excitation or inhibition.

2.2. The C-discriminator

Figure 2 shows the structure of a C-discriminator. It is an array of RAMs whose memory locations can hold a continuous range of values between 0 and 1. The number K of such RAMs in a C-discriminator is

$$K = \frac{m}{n},$$

n being the n -tuple size and m the number of inputs to the C-discriminator. The value m is a multiple of n . The output ϕ_κ of a RAM κ takes real values between 0 and 1. We denote by

$$i_\kappa = (i_1, \dots, i_n)_\kappa$$

the binary vector representing both the input address to RAM κ and the storage location addressed by it. The value stored in a memory location with address $\mathbf{a} = (a_1, \dots, a_n)$ is denoted $[a]$ or $[a_1, \dots, a_n]$. Then, it holds that

$$\phi_\kappa = [i_1, \dots, i_n]_\kappa.$$

The output ρ_γ of a discriminator γ can be written as

$$\rho_\gamma = \frac{1}{K} \sum_{\kappa=0}^K \phi_\kappa = \frac{1}{K} \sum_{\kappa=0}^K [i_1, \dots, i_n]_\kappa^\gamma,$$

with $0 < \rho_\gamma < 1$. Hereafter ρ_γ will be quoted as a percentage.

One may notice that the structure of the C-discriminator is identical to that of the linear

neuron model, in the limiting case $n=1$. Furthermore, the C-discriminator structure can be viewed as a intermediate case between a RAM ($n=m$ and $K=1$) and a linear neuron ($K=m$ and $n=1$).

2.3. An unsupervised training algorithm

The principle of the training procedure follows the rules that form the basis of various self-organizing processes. When a pattern is presented to the net, the best matching neuron is located, that is the one which responds with the highest value. Then, the matching is either increased at this neuron and its topological neighbours or decreased at more distant neurons. We proceed by examining this procedure in detail.

First, the memory content of the neurons is initialised at random values within the interval $[0,1]$. Then, the net is presented with an input pattern chosen randomly amongst a set of training patterns. All the C-discriminators see that pattern simultaneously. The neuron Φ whose response ρ_Φ to the pattern is the highest is said to be *firing*. That means that this neuron is the one which matches best the input pattern and it holds that

$$\rho_\Phi = \max(\rho_\gamma), \quad \text{with } \gamma = 1, \dots, Q.$$

We define one excitation and one inhibition region of interaction about neuron Φ . The spatial extents of these regions are characterised by an excitation radius r_{exc} and an inhibition radius r_{inh} respectively, as shown on Figure 3. The memory content of the neurons in these regions is updated in such a way that the neurons lying in the excitation region become more susceptible to fire when the same or a similar pattern is presented again to the net, while the neurons in the inhibition region are 'pushed away' in terms of their response to the input pattern. All other neurons in the net are left unchanged.

Consider a particular RAM R_λ in a C-discriminator situated in the excitation region of neuron Φ . We denote by H_λ the Hamming distance between the input i_λ to R_λ and the address a_λ of a certain memory location to be updated in R_λ . Every memory location a_λ in R_λ will be updated according to the relation

$$[a_\lambda]' = [a_\lambda] + k|\Delta|(\tau(\Delta) - [a_\lambda]), \quad \text{with } \Delta = \frac{n}{2} - H_\lambda, \quad (1)$$

τ being the Heaviside function defined as

$$\tau(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

$|\Delta|$ being the absolute value of Δ , $'$ denoting an updated value, and k being an *excitation coefficient*, with $0 < k < 1$. Similarly for a neuron in the inhibition region,

$$[a_\lambda]' = [a_\lambda] + l|\Delta|(\tau(-\Delta) - [a_\lambda]),$$

l being an *inhibition coefficient*, with $0 < l \ll k$. In many simulations the neural density is insufficient to include a region of inhibition. The general nature of the process is similar for a wide range of values k and l . It is mainly the speed of the process which varies.

Following the updating phase, another input pattern is selected and presented to the net, and the training process starts again. The number of training iterations is problem dependent and can be as small as 1000, as in the example presented in this paper. One of the most central phenomena associated with this training procedure is a clustering effect that makes similar patterns to produce output responses close to one another in the network output map.

3. EXPERIMENTAL RESULTS

3.1. The simulation

The problem given to the net is to locate the position, in the input pattern, of a black area, here represented as a square. The training patterns form four classes. In each of them, the black square is positioned at one of the corners of the patterns. Patterns belonging to the same class differ by a certain number of pixels chosen randomly and whose color has been

inverted to the opposite one. This acts as an addition of noise to the four basic patterns. In the simulation, up to 4 pixels were flipped in an input pattern. Each input pattern consists of an array of 8 by 8 pixels. Therefore there are 64 inputs fed into each of the net C-discriminators. The net is made of 40 C-discriminators arranged in a one-dimensional array. This yields a one-dimensional map of responses. The size of the n-tuple is chosen to be 4. Consequently, there are 16 RAMs in each C-discriminator.

The inhibition interactions between neurons are neglected. The radius of excitation is treated as a decreasing function of the number of input patterns seen, starting at a value of 10 and decreasing by 1 after each 100 training steps. Although this decreasing radius improves moderately the response clusterings, this is not a critical factor in the behaviour of the net, compared to a fixed radius. It may be argued that decreasing the radius of the excitation region, as learning proceeds, is somewhat analogous to the process of annealing in Boltzmann nets. The excitation coefficient is chosen equal to 0.1.

In Figure 4, we show the net responses after 1000 training steps. 9 test patterns are chosen, labelled $P_1, P_2, \dots, P_9$. Patterns P_1 to P_4 are seen by the net during training, whereas patterns P_5 to P_9 are intermediate patterns never seen before.

Although it is difficult to derive mathematically the responses given by the net starting from equation (1), we shall try to justify them considering the principle of "overlap areas" on which the WISARD system is based [1]. In the following, we do not consider the discriminators situated in areas of transitional responses. We first group all other discriminators in four sets $S_1, \dots, S_4$ such that discriminators belonging to set S_i give 100 % response when presented with the pattern P_i . Then, the response $\rho_{S_i}(P_j)$ of a discriminator, belonging to set S_i , to a pattern P_j is given by

$$\rho_{S_i}(P_j) = \text{overl.area}(P_i, P_j),$$

defining $\text{overl.area}(P_i, P_j)$ as the percentage of identical pixels in patterns P_i and P_j . The overlap area between each pairwise combination of the patterns P_1 to P_4 is 50 %, observed to be a reasonable prediction of the responses obtained in the simulation results of Figure 4. Considering the case of an intermediate pattern, for example P_5 , we observe that it has 75 % overlap area with patterns P_1 and P_2 , and 50 % with patterns P_3 and P_4 . Pattern P_9 produces 62.5 % response at each C-discriminator, which is the overlap area between P_9 and each of the patterns P_1 to P_4 .

3.2. Temporal evolution of responses

Figure 5 shows the temporal evolution of neuron responses to a certain pattern at different times throughout the training procedure. Initially, the C-discriminator which outputs the strongest response for a particular pattern is randomly distributed in the net. As learning proceeds, a 'bubble' of response begins to emerge in a certain region of the output map. Then, the clustering of activity stabilises to produce a stable output map of responses.

4. CONCLUSIONS

A self-organizing network similar in structure to a Kohonen net has been presented. The C-discriminator neuron model used is an intermediate case between the RAM model and the linear one. The main thrust of this paper is to show that such a model can be used in a network governed by an unsupervised training procedure. The simulations have shown that during training, a clustering of responses to similar patterns appears, resulting in a topologically ordered map. It is argued that the output responses of the neurons can be predicted on the basis of overlap areas between patterns.

REFERENCES

- [1] Aleksander, I., Thomas, W. V., and Bowden, P. A., 1984, "WISARD, a radical step forward in image recognition", *Sensor Review*, vol.4, no 3, pp. 120-124.
- [2] Kohonen, T., 1988, "Representation of Sensory Information in Self-organizing feature

- 5 -

- maps, and the relation of these maps to distributed memory networks", *Computer Simulation in Brain Science*, Ed. Rodney M. J. Cotterill, Cambridge University Press.
- [3] Kohonen, T., 1989, *Self-Organization and Associative Memory*, 3th edition, Springer-Verlag, New York.
- [4] Bledsoe, W. W. and Browning, I., 1959, "Pattern Recognition and Reading by Machine", *Proc. Eastern Joint Computer Conf. Boston, Mass.*.

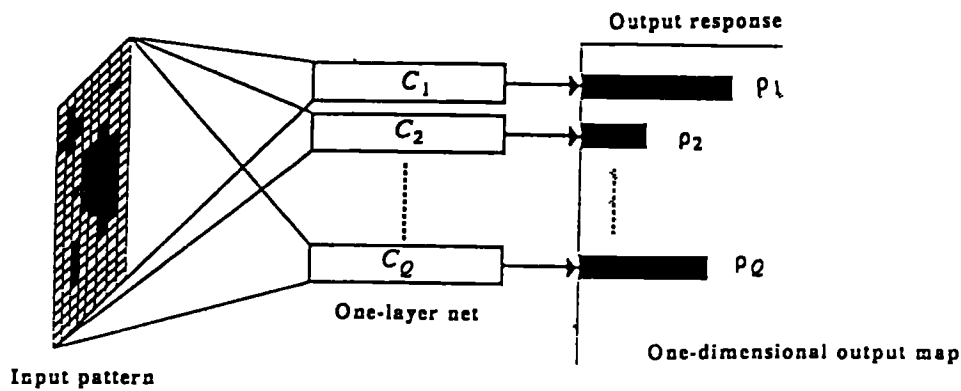


Figure 1 : The C-discriminator-based neural network.

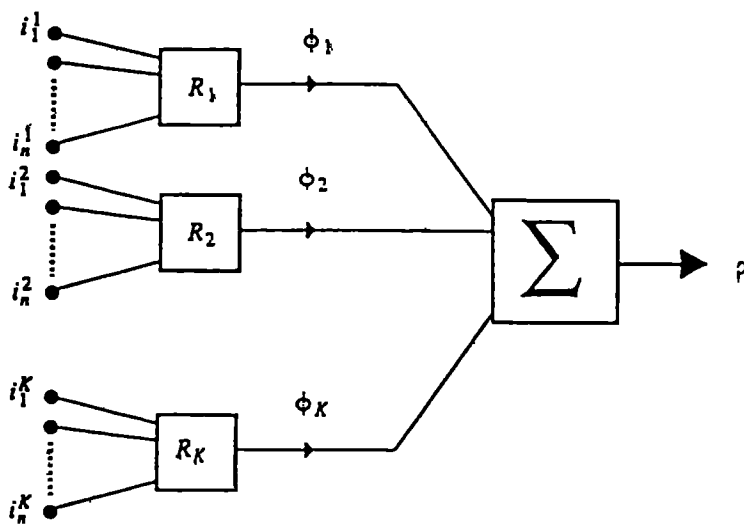
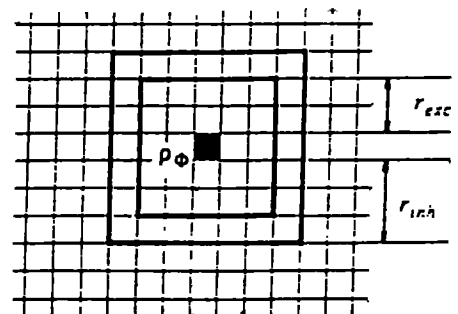


Figure 2 : Structure of a C-discriminator

Figure 3 : Excitation and inhibition zones about the firing neuron.
 $r_{exc} = 2 \times r_{inh} = 3$.

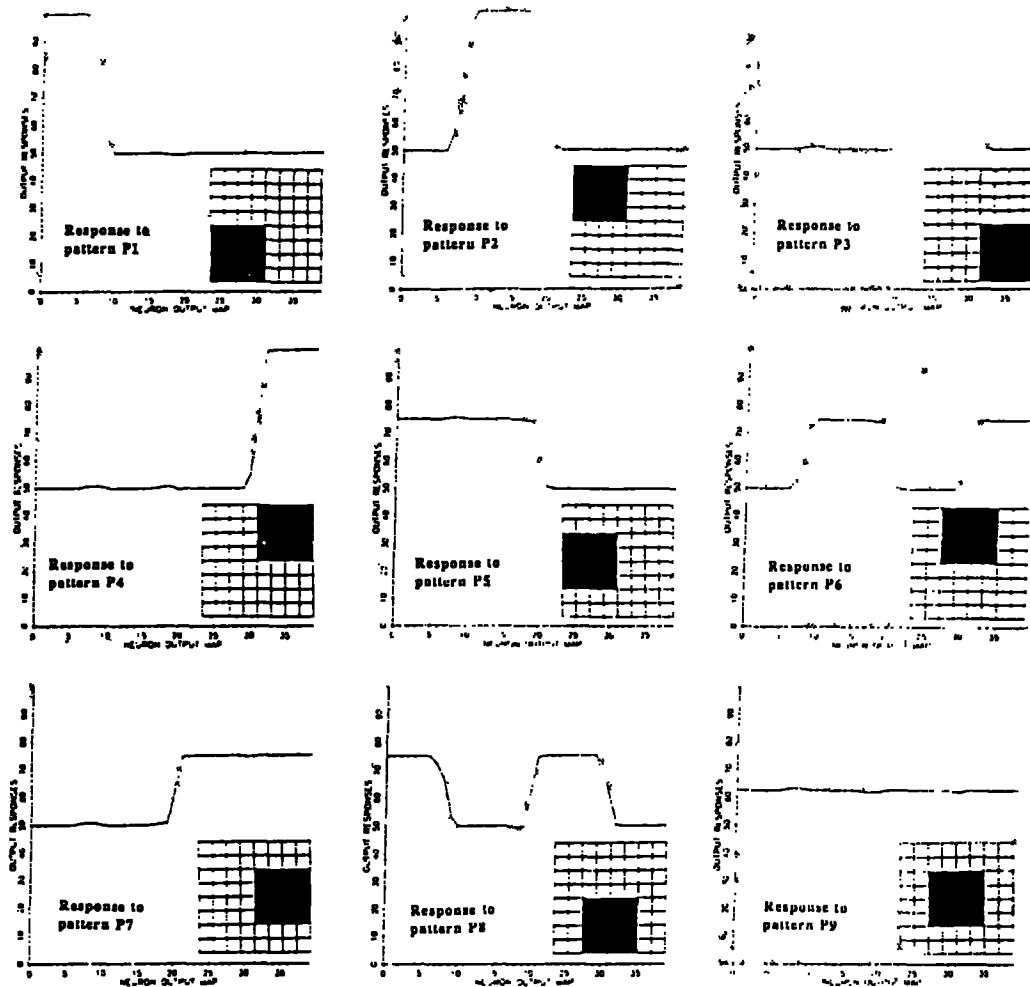


Figure 4 : Responses to the test patterns P1 to P9 at time T=1000.

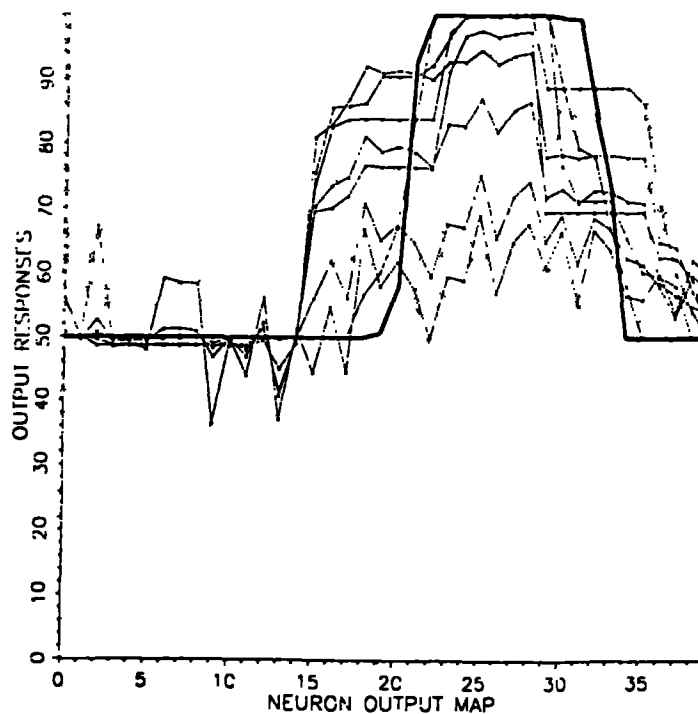


Figure 5 : Development of responses to a particular pattern over time. The figure shows the responses at times T=0, 10, 20, 30, 50, 100, 200, 500, and 1000.

STORAGE CAPACITY AND RETRIEVAL PROPERTIES OF AN AUTO-ASSOCIATIVE GENERAL NEURAL UNIT

Panayotis Ntounoufis

Neural Systems Engineering Laboratory
Imperial College, London SW7 2BT, England
email: JANET panos%winge@sig.ee.ic.ac.uk

ABSTRACT

The probability of disruption of patterns stored in a general neural unit, used as an auto-associator, is derived. It is shown that the network can store about the same number of patterns as the number of inputs per node, with an acceptable error rate. The retrieval equations of the network are established in the case of three arbitrary stored patterns.

1. INTRODUCTION

A cluster of interconnected formal neurons has, as an emergent property, the ability to enter stable firing patterns, when stimulated by the presentation of noisy versions of these patterns [1]. In that case, the network is said to perform auto-association. In this paper, this property is studied using a network called General Neural Unit, or GNU [3],[4]. A schematic diagram of this network is shown in Figure 1. It consists of one layer of K processing elements or nodes. The inputs to a node consist of N external inputs and a proportion Q of the K outputs of the system. Then, the total number of inputs to a node is given by $N + F$, where $F = QK$ is the number of feedback connections to a node. When $F = 0$, the system operation is that of a feed-forward network. In this paper, we shall be concerned with the case $N = 0$ and $F \neq 0$, so that the GNU is used as an auto-associator.

The nodes of the GNU are GRAMs (Generalising Random Access Memories)[2],[3]. The input to a node represents the address of a GRAM. When a node is required to associate an input pattern with an output value, then this value is stored at the memory location addressed by the input to the node. Memory locations can store the values 0, 1, or μ . The value μ stands for "unknown" and, when it is addressed, the node outputs 0 or 1 with equal probability. Before any training takes place in the GNU, all memory locations in all nodes are set to the μ value. We can now examine the different phases in the processing of the GRAM. We distinguish three phases. The *learning phase* is the operation by which the node records addresses with their required response. During the *operating phase*, the node makes use of its stored information. There is an additional phase, called *spreading*, during which memory locations not addressed during the learning phase, are affected by the use of a suitable algorithm [3]. The aim of this phase is to diffuse the information acquired by the node during training in order to be able to generalise its knowledge to "unseen" patterns. Different spreading procedures can be used. Here, the spreading proceeds as follows. We consider in turn all memory locations in a node. If a memory location contains 1 or 0, it is left unchanged. If it contains μ , it is set to 1 (or 0) only if its address is closest in Hamming distance to the address of a memory location storing a 1 (or 0); otherwise it is left unchanged. After this operation, the node is said to have *full generalisation*, that is, unless there is a contradiction, all memory locations in the node are set either to 1 or 0.

2. PROBABILITY OF DISRUPTION AND STORAGE CAPACITY

In this section, we are concerned with a weakly interconnected auto-associator ($N = 0$, no external inputs to a node), that is, the number of input lines F to a node is much smaller than the number of nodes K in the network. We consider a set of S training patterns $T_1, T_2, \dots, T_S$ to be stored in the system. For simplicity, the overlap between any two patterns is assumed to be equal and minimal. Thus,

$$\text{overlap}(T_i, T_j) = 1 - \frac{2}{S} \quad (1)$$

One graphical arrangement of such patterns might be to represent an input pattern as divided into S sections $s_1, s_2, \dots, s_S$ of equal area. Section s_j , corresponding to pattern T_j , contains 1-values and all other sections contain 0-values. During the storage process, contradictions occur when new stored patterns cause old ones to no longer be stable patterns in the network. The object of what follows is the calculation of the probability of disruption of pattern T_1 by patterns $T_2, \dots, T_S$.

We consider first the probability of pattern T_1 being disrupted by pattern T_2 . This occurs when the input samples to a particular node are identical for patterns T_1 and T_2 , while the required outputs are different. In other words, a contradiction occurs because the node is required to store different values at the same memory location. This probability is

$$Pr(T_1 \text{ disrupted by } T_2) = \frac{2}{S} \left(1 - \frac{2}{S}\right)^F.$$

$\frac{2}{S}$ is the probability of having different outputs for T_1 and T_2 , while $\left(1 - \frac{2}{S}\right)^F$ is the probability that all inputs to the node are the same for T_1 and T_2 .

We proceed further and consider now the probability that pattern T_1 is disrupted by all other patterns $T_2, \dots, T_S$. We denote by O_j the output required for pattern T_j , and by I_j the inputs for pattern T_j . All different cases of contradictions need to be considered.

First, we consider the case where $O_1 \neq O_j$, for any j such that $j \neq 1$. This case has a probability $\frac{1}{S}$, taking into account (1). For the inputs, we have to consider the probability that I_1 is equal to at least another I_j , with $j \neq 1$. This probability is

$$1 - \left[1 - \left(1 - \frac{2}{S}\right)^F\right]^{S-1}.$$

This provides the first term of the overall probability of disruption:

$$\frac{1}{S} \left[1 - \left[1 - \left(1 - \frac{2}{S}\right)^F\right]^{S-1}\right].$$

Then, we consider the $S-1$ cases ($k=2, \dots, S$) for each of which $O_1 \neq O_k$ and $O_1 = O_j$, for any j such that $j \neq k$ and $j \neq 1$. These cases have each a probability of $\frac{1}{S}$. For the inputs, we have to consider the probability that $I_1 = I_k$ which is $\left(1 - \frac{2}{S}\right)^F$. This gives us the second term in the overall probability of disruption:

$$\frac{S-1}{S} \left(1 - \frac{2}{S}\right)^F.$$

Finally, we obtain the overall probability of disruption $P_d(S, F)$:

$$P_d(S, F) = \frac{1}{S} \left[1 - \left[1 - \left(1 - \frac{2}{S}\right)^F\right]^{S-1}\right] + \frac{S-1}{S} \left(1 - \frac{2}{S}\right)^F. \quad (2)$$

Hence, $P_d(S, F)$ is the probability of disruption of pattern T_1 , in a particular node, due to patterns $T_2, \dots, T_S$. Therefore, on average, a proportion $P_d(S, F)$ of nodes exists where pattern T_1 is disrupted, and the law of large numbers, with respect to K , yields that $P_d(S, F)$ is the probability that pattern T_1 is disrupted in the whole network. Finally, $P_d(S, F)$ may be interpreted as the proportion of patterns that are disrupted in the network.

It is interesting to consider the following limit cases:

If $S=2$, then $P_d(S, F)=0$. It is indeed obvious that any GNU can store two orthogonal patterns without disruption, because at each node, the patterns will address a different memory location.

For large S , the probability of disruption tends to 1:

$$\lim_{S \rightarrow \infty} P_d(S, F) = 1.$$

If S is held to the same value as F , the probability of disruption is given by

$$\lim_{S=F-\infty} P_d(S,F) = \frac{1}{e^2}.$$

Thus 13.53 % of the total number of patterns are disrupted in the above case. This shows that the network can store about the same number of patterns as the number of inputs per node, with an acceptable error probability.

The main conclusion is that the proportion of disrupted patterns, and therefore the storage capacity of the network, depend primarily on the number of signals F that a node samples. Similar results are reported by other authors [4],[5],[6] and this property seems to be general in RAM-based systems.

3. RETRIEVAL EQUATIONS

3.1. Retrieval of two orthogonal patterns

We consider a GNU trained on two orthogonal patterns T_1 and T_2 . We assume that a spreading process, with full generalisation, takes place in the network nodes after training. At time t , the GNU is in a state that has an overlap $x(t)$ with pattern T_1 , and therefore an overlap $1 - x(t)$ with pattern T_2 . This is extensively studied in [4]. We just state here the retrieval equation of the system:

$$x(t+1) = f(x(t), F) = \begin{cases} \sum_{j=\lceil \frac{F}{2} \rceil}^F \binom{F}{j} x(t)^j (1-x(t))^{F-j} & F \text{ odd} \\ \sum_{j=\lceil \frac{F}{2} \rceil}^F \binom{F}{j} x(t)^j (1-x(t))^{F-j} + \frac{1}{2} \binom{F}{\frac{F}{2}} x(t)^{\frac{F}{2}} (1-x(t))^{\frac{F}{2}} & F \text{ even,} \end{cases}$$

where $\lceil x \rceil$ is the ceiling of x . In [4], it is shown that the system enters the trained state which has the greater overlap with the initial state of the GNU. The larger the number of inputs F , the more rapidly the system will converge.

3.2. General retrieval of three patterns

In this subsection, we establish the equations governing the retrieval process of the system in the case of three stored patterns. Again, the network is trained with three patterns T_1, T_2, T_3 . Then, a spreading phase, with full generalisation, takes place in the nodes. An unknown pattern U is presented to the network and the evolution through time of the variables $x_1(t), x_2(t)$, and $x_3(t)$ is monitored. Those variables represent the overlaps, at time t , between pattern U and patterns T_1, T_2 , and T_3 , respectively. We denote by ω_{ij} the overlap between trained patterns T_i and T_j . Considering a particular F -input node, we denote by n_j the number of inputs that have the same values for U and T_j .

In Figure 2(a), the patterns T_1, T_2, T_3 are represented as sets in a Venn diagram. In each region the relative proportions of pixels that belong to that region, or equivalently the probabilities that an input or output to a node takes its value from that region, are indicated. For example, if corresponding pixels in patterns T_1, T_2, T_3 are compared, α is the proportion of pixels in pattern T_1 that have opposite values in patterns T_2 and T_3 (Fig. 2). We write $\alpha, \beta, \gamma, \delta$ as functions of the overlaps ω_{ij} between training patterns, or equivalently as functions of their distances $d_{ij} = 1 - \omega_{ij}$:

$$\begin{aligned}
\alpha &= \frac{1}{2} (d_{12} + d_{13} - d_{23}) \\
\beta &= \frac{1}{2} (-d_{12} + d_{13} + d_{23}) \\
\gamma &= \frac{1}{2} (d_{12} - d_{13} + d_{23}) \\
\delta &= 1 + \frac{1}{2} (-d_{12} - d_{13} - d_{23})
\end{aligned} \tag{3}$$

with $\alpha + \beta + \gamma + \delta = 1$.

In Figure 2(b), the areas of intersection of patterns T_1, T_2, T_3 with pattern U are shown. This defines the proportion variables ξ, η, θ , and λ . For example, a proportion ξ of the pixels in region α will be equal for patterns T_1 and U , and so a proportion $1-\xi$ will be equal for patterns T_2, T_3 , and U . This holds true for the other regions. Dropping for the time being the dependence on t , it is straightforward to write the relations

$$\begin{aligned}
x_1 &= \xi\alpha + (1-\theta)\beta + (1-\eta)\gamma + \lambda\delta \\
x_2 &= (1-\xi)\alpha + (1-\theta)\beta + \eta\gamma + \lambda\delta \\
x_3 &= (1-\xi)\alpha + \theta\beta + (1-\eta)\gamma + \lambda\delta
\end{aligned} \tag{4}$$

When a sample of pattern U is presented to a particular node of the GNU, the node will output a value corresponding to the training of pattern T_k , if $n_k = \max(n_j)$, for $j=1, 2$, or 3 . For example, if the input corresponding to pattern U is closest to that corresponding to pattern T_1 , then the output will be the same as for pattern T_1 . This results from the spreading phase of the learning algorithm. There are 7 different cases to be considered, whose probabilities of occurrence $p_1, \dots, p_7$ are shown below:

$$\begin{aligned}
p_1 &= \text{Pr}[(n_1 > n_2) \text{ and } (n_1 > n_3)] \\
p_2 &= \text{Pr}[(n_2 > n_1) \text{ and } (n_2 > n_3)] \\
p_3 &= \text{Pr}[(n_3 > n_1) \text{ and } (n_3 > n_2)] \\
p_4 &= \text{Pr}[(n_1 = n_2) \text{ and } (n_3 < n_1)] \\
p_5 &= \text{Pr}[(n_1 = n_3) \text{ and } (n_2 < n_1)] \\
p_6 &= \text{Pr}[(n_2 = n_3) \text{ and } (n_1 < n_2)] \\
p_7 &= \text{Pr}[(n_1 = n_2) \text{ and } (n_1 = n_3)]
\end{aligned} \tag{5}$$

with $\sum_i p_i = 1$. The exact formulas yielding the p_i 's are sums over a multinomial distribution whose parameters are the variables associated with the different regions represented in figure 2(b). That is, for example,

$$\begin{aligned}
p_1 = \sum_{\substack{m_1+m_6 > m_2+m_3 \\ m_1+m_4 > m_2+m_3}} \frac{F!}{m_1! m_2! \dots m_8!} [\alpha\xi]^{m_1} [\alpha(1-\xi)]^{m_2} [\beta\theta]^{m_3} [\beta(1-\theta)]^{m_4} \\
[\gamma\eta]^{m_5} [\gamma(1-\eta)]^{m_6} [\delta\lambda]^{m_7} [\delta(1-\lambda)]^{m_8}
\end{aligned}$$

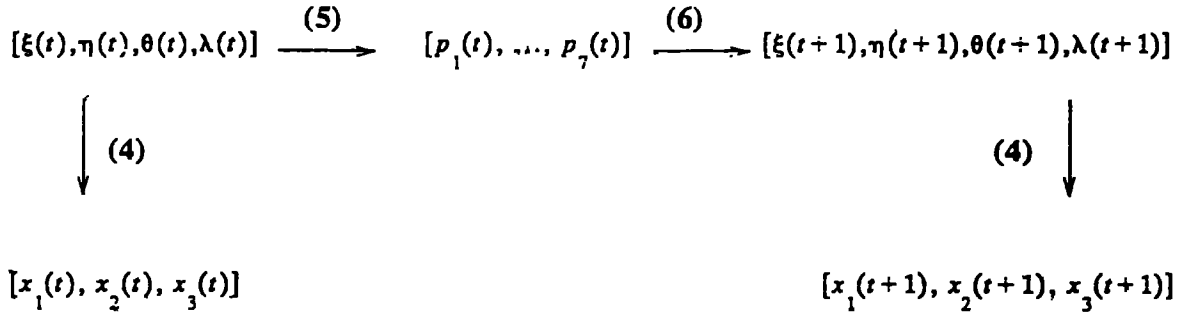
We express the variables x_1, x_2, x_3 as functions of the p_i 's. For example,

$$\begin{aligned}
x_1 &= p_1(\alpha + \beta + \gamma + \delta) + p_2(\beta + \delta) + p_3(\gamma + \delta) + p_4(\beta + \delta + \frac{1}{2}\alpha + \frac{1}{2}\gamma) \\
&+ p_5(\gamma + \delta + \frac{1}{2}\alpha + \frac{1}{2}\beta) + p_6(\delta + \frac{1}{2}\beta + \frac{1}{2}\gamma) + p_7(\delta + \frac{1}{2}\alpha + \frac{1}{2}\beta + \frac{1}{2}\gamma)
\end{aligned}$$

We obtain similar expressions for x_2 and x_3 . Then, using these expressions in conjunction with (4), the values of ξ, η, θ , and λ can be identified and their evolution in time is given by:

$$\begin{aligned}
\xi(t+1) &= p_1(t) + \frac{1}{2}(p_4(t) + p_5(t) + p_7(t)) \\
\eta(t+1) &= p_2(t) + \frac{1}{2}(p_4(t) + p_6(t) + p_7(t)) \\
\theta(t+1) &= p_3(t) + \frac{1}{2}(p_5(t) + p_6(t) + p_7(t)) \\
\lambda(t+1) &= 1,
\end{aligned} \tag{6}$$

These expressions, when substituted in (4), yield the retrieval equations of the system. The interdependence among the sets of variables, determining the system evolution in time, can be best understood from the simple flow chart shown below:



The retrieval equations are a useful tool for studying the dynamics of the system, the stable states, and the extent of the basins of attraction.

4. CONCLUSION

The probability of disruption of patterns stored in a general neural unit, used as an auto-associator, has been calculated. It was shown that the network can store about the same number of patterns as the number of inputs per node, with a tolerable error level. The retrieval equations of the network have been established in the case of three arbitrary stored patterns. These equations govern the dynamics of the system.

REFERENCES

- [1] Hopfield, J. J., "Neural Networks and Physical Systems with Emergent Collective Properties", *Proc. Nat. Acad. Sci. USA*, 79, 2554-58, 1982.
- [2] Aleksander, I., "Ideal Neurons for Neural Computers", *Proc. Int. Conf. on Parallel Processing in Neural Systems and Computers*, Düsseldorf, Springer Verlag, 1990.
- [3] Aleksander, I., and Morton, H. B., "An Overview of Weightless Neural Systems", *Proc. IJCNN-90*, Washington, D. C., 1990.
- [4] Aleksander, I., "Weightless Neural Tools: Towards Cognitive Macrostructures", *Int. Rep. NSEIR/IA# 1/90*, Oct. 1990.
- [5] Wong, K.Y.M., and Sherrington, D., "Storage Properties of Boolean Neural Nets", *Proc. NEURO88*, Paris, 1988.
- [6] Wong, K.Y.M., and Sherrington, D., "Theory of Associative Memory in Randomly Connected Boolean Neural Networks", *J. Phys. A: Math. Gen.*, 22, pp. 2233-63, 1989.

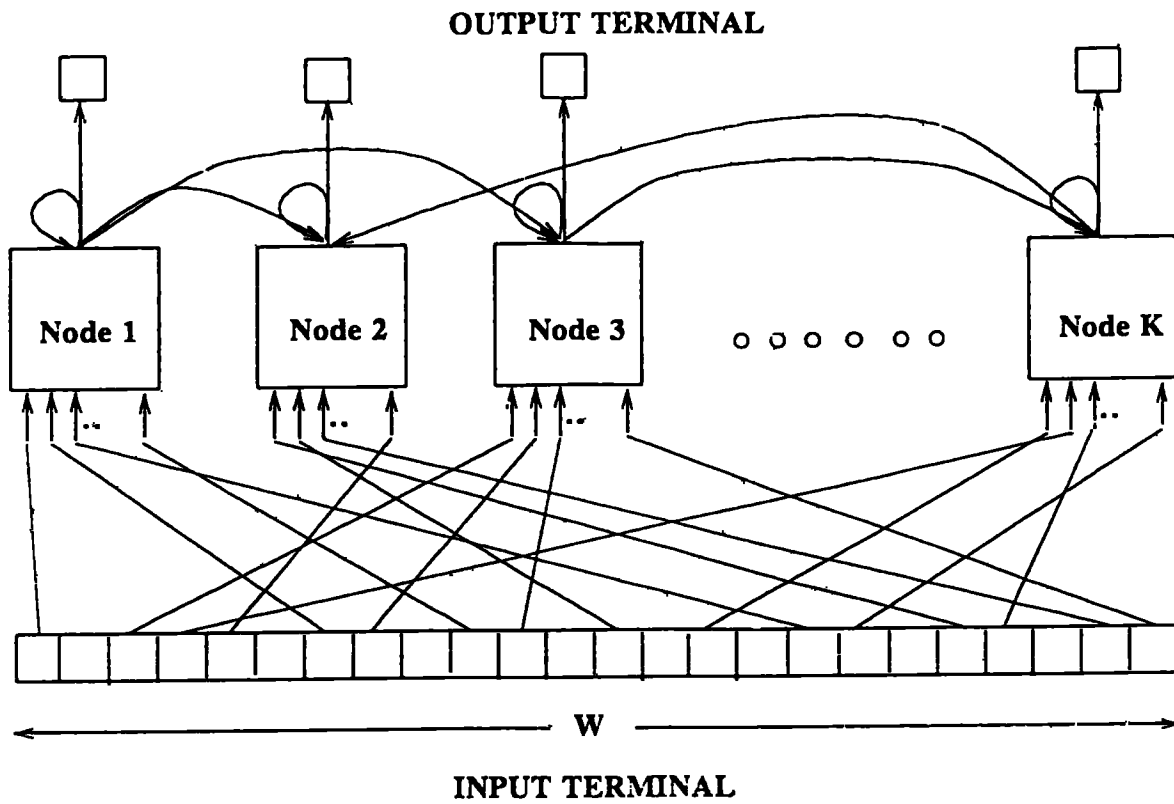


Figure 1. General neural unit.

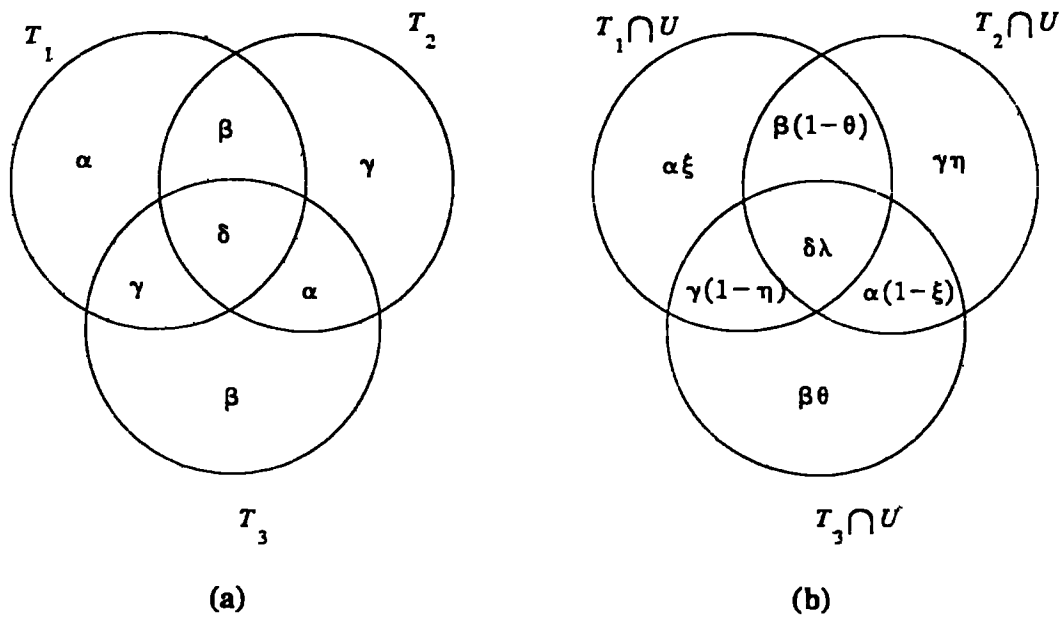


Figure 2. Patterns are represented in a Venn diagram in which the proportional areas associated with each region are indicated.

PARALLEL HARDWARE IMPLEMENTATION OF THE GENERAL NEURAL UNIT MODEL

P. Ntourtoufis

Neural Systems Engineering Laboratory, Department of Electrical and Electronic Engineering, Imperial College of Science, Technology, and Medicine, London SW7 2BT, England

Abstract

This paper presents a parallel hardware implementation of the General Neural Unit model. The proposed architecture is a MIMD machine consisting of a series of rings connecting a number of processors together. The rings are themselves connected to each other through an additional ring. This scheme involves a compromise between a ring architecture and a two-dimensional mesh.

1. BACKGROUND AND MOTIVATION

A neural network called the General Neural Unit [1][2], or GNU, was studied by Aleksander [3] and Ntourtoufis [4] to investigate its ability to perform auto-association [5]. It consists of one layer of K processing elements or nodes. The inputs to a node consist of N external inputs and F feed-back inputs. The feed-back inputs are chosen randomly from the K outputs of the system. The nodes of the GNU are GRAMs (Generalising Random Access Memories) [1]. The input to a node represents the address of a GRAM. When a node is required to associate an input pattern with an output value, then this value is stored at the memory location addressed by the input to the node. There are three different phases in the processing of the GRAM. The *train* phase is the operation by which the node records addresses with their required response. During the *use* phase, the node makes use of its stored information. There is an additional *spread* phase, during which memory locations not addressed during the train phase, are affected by the use of a suitable algorithm [3]. The aim of this phase is to diffuse the information acquired by the node during training in order to be able to generalise its knowledge to "unseen" patterns. Different spreading procedures can be used and a degree of generalisation G can be defined.

The aim of this paper is to present a new hardware architecture implementing the GNU model.

2. PARALLEL IMPLEMENTATION

2.1. Specifications and requirements

A typical example of what is expected to be implemented on a printed board is that of 1024 GRAMs, with 8 inputs per GRAM. The GRAMs contain 2-bit words, in order to code the 3 different values that a memory location can hold [2]. Variable level of

feed-back F and variable degree of generalisation G are allowed. Amongst the important operations performed by the system are *clamp*, *run*, and *spread*, which correspond to the train, use, and spread phases of a GRAM, respectively.

The amount of memory required to implement 1024 8-input GRAMs is

$$1024 \text{ GRAMs} \cdot 256 \text{ memory locations} \cdot 2 \text{ bits} = 0.5 \text{ Mbits.}$$

A fundamental aspect of the requirements of the implementation concerns the desired connectivity. One should be able to connect any of the GNU's external inputs, to any input of any GRAM. Likewise, one should be able to connect any of the GNU's outputs, to any input of any GRAM. These constraints will justify the choices made in the design of the information transmission mechanism from the GNU's external terminal to the GRAMs, and between GRAMs.

2.2. General architecture

Figure 1 shows the general architecture proposed here. It consists of a *major* ring and 8 *minor* rings.

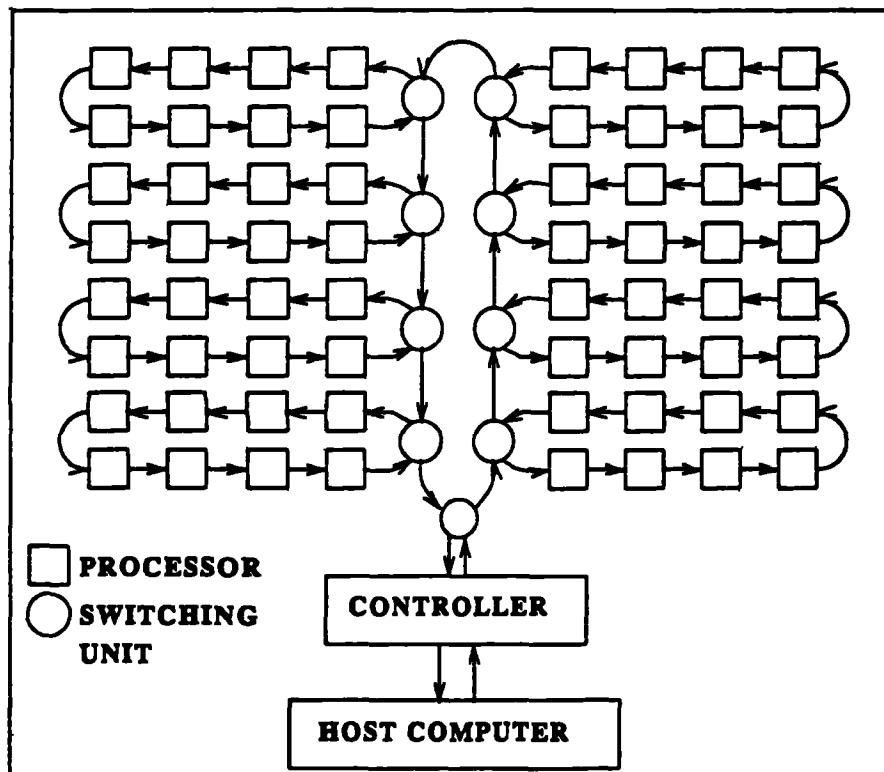


Figure 1: Hardware architecture

The major ring connects 9 *switching units* together. Each minor ring connects 8 *processors* together and is connected to the major ring through one the switching units. There are therefore 64 processors in the system and each is implemented as a VLSI chip. A processor implements the functionality *and* memory of a number of GRAMs (typically 16 GRAMs). This number can vary according to the number of inputs per GRAM and therefore the memory required for each GRAM.

This architecture can be classified as that of a *MIMD* (multiple instructions, multiple data) machine. The nodes of a MIMD machine support one or more independent

processes each. Within the class of MIMD machines it is possible to classify our architecture further, as a *network* architecture. In a network architecture the switches are distributed throughout the machine, and if there are no switches between the processors these are directly connected [6]. Finally this architecture can be seen as a *mesh* architecture where a compromise between a one- and two-dimensional grid has been made.

2.3. Messages

Communication between processors consists of messages. A message, once issued by a *source* processor, is passed from processor to processor, until it reaches its *destination* processor. The message takes the shortest possible route to travel. It goes first along the minor ring to which the source processor belongs, then travels along the major ring, and finally along the minor ring to which the destination processor belongs.

A message consists of about 20 bits which are divided in three fields: the *address field* (address of the destination processor and address of the destination GRAM), the *command field*, and the *datum field*. In each processor there is a router that uses part of the address field to route the message either to the processor itself or to its neighbour. The routing of messages is kept to its simplest form, unlike architectures such as the Connection machine [7]. Typical messages include the transmission of an input bit, the transmission of the required output bit, addressing commands, storage commands, and spreading commands.

2.4. Organisation of processor

Each processor is implemented in a single VLSI integrated circuit. The silicon memory storing the GRAM's contents is implemented on chip. It is made of one or more RAM blocks, each providing the memory for a certain number of GRAMs. Figure 2 shows a block diagram of a processor. For example, each RAM block may consist of 256 words of 32 bits each. Different uses of the memory bits are possible. It is possible for example to implement 256 4-input GRAMs, or 16 8-input GRAMs. Thus, there is a trade-off between the number of GRAMs and the number of inputs per GRAM.

2.5. Timing considerations

A very important parameter in MIMD machines is the relative speed of computation and communication. This is currently being investigated.

3. DISCUSSION

We discuss here the choices made in the design of the architecture described in the previous section.

Unidirectional connections are used to avoid traffic jams of messages. Indeed these can occur if two messages attempt to move in opposite directions and use the same connection bus. This scheme enables a very simple message routing in the processor. Still, it might happen that, for example, two messages are attempting to enter the same switching unit, one entering the ring and one already circulating in the ring. Therefore, a system of priorities is installed: a message already in a ring has priority over a message entering that ring.

The option of a one-dimensional grid, or ring, has been discarded in order to keep the communication time, bottleneck of most MIMD machines, as short as possible.

The reason for not choosing a 2-dimensional grid is twofold. First, the routing of

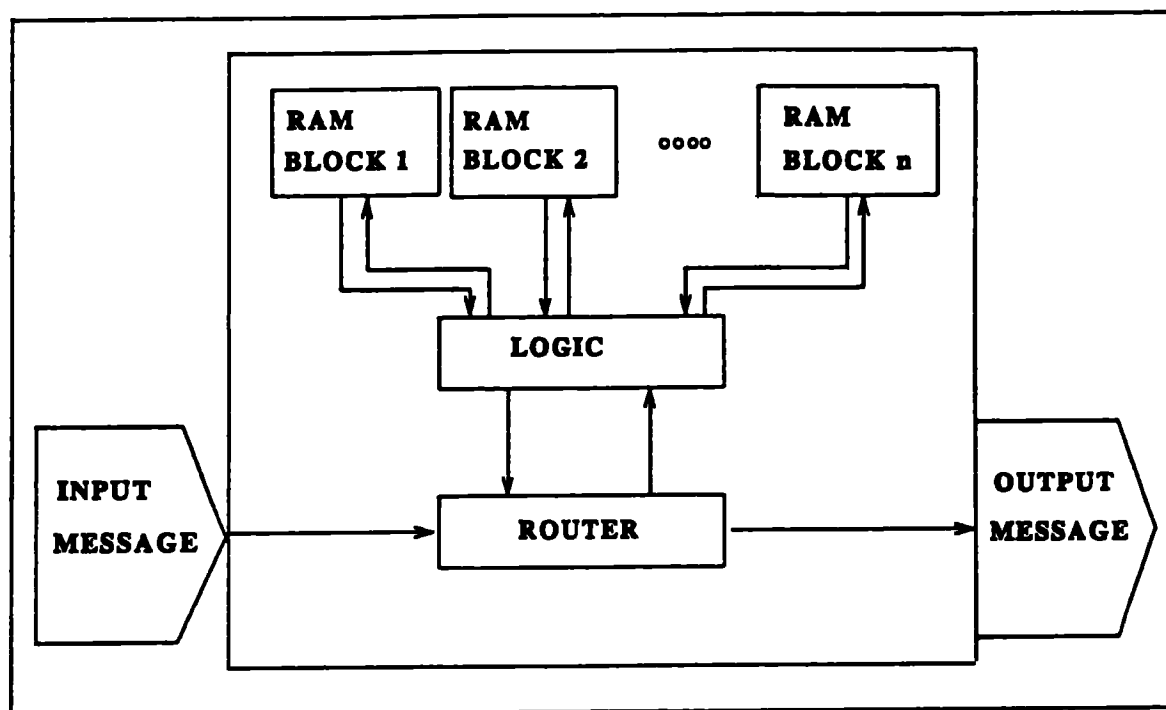


Figure 2: Block diagram of a processor

messages has to be kept simple. Second, it is necessary to have only one input message and one output message per processor in order to keep the number of chip input pins small, taking into account the fact that a message goes from one chip to the next in a single clock cycle (parallel transmission), this meaning about 20 input message pins and 20 output message pins.

4. REFERENCES

- 1 Aleksander, I., "Ideal Neurons for Neural Computers", *Proc. Int. Conf. on Parallel Processing in Neural Systems and Computers*, Düsseldorf, Springer Verlag, 1990.
- 2 Aleksander, I., and Morton, H. B., "An Overview of Weightless Neural Systems", *Proc. IJCNN-90*, Washington, D. C., 1990.
- 3 Aleksander, I., "Weightless Neural Tools: Towards Cognitive Macrostructures", *The CAIP Neural Network Workshop*, Rutgers University, New Jersey, October 1990.
- 4 Ntourtoufis, P., "Storage Capacity and Retrieval Properties of an Auto-associative General Neural Unit", *Proc. IJCNN-91*, Seattle, 1991.
- 5 Hopfield, J. J., "Neural Networks and Physical Systems with Emergent Collective Properties", *Proc. Nat. Acad. Sci. USA*, 79, 2554-58, 1982.
- 6 Másson, E., "A Parallel Neural Network Simulator trained for Image Compression", Nordita preprint, September 1990.
- 7 Hillis, W. D., *The Connection Machine*, MIT Press, 1985.

A DYNAMICALLY GENERALISING WEIGHTLESS NEURAL ELEMENT

P. Ntourtoufis

Neural Systems Engineering Laboratory, Department of Electrical and Electronic Engineering, Imperial College of Science, Technology and Medicine, London SW7 2BT, U.K.

Abstract

The paper introduces the concept of the *Dynamically Generalising* weightless Neuron (DGN). The DGN is derived from the Generalising Random Access Memory [1], in which the *training* and *spreading* operations are replaced by a single *learning* phase. The DGN is able to store and spread patterns, through a dynamical process involving interactions between each memory location and its immediate neighbours, and external signals. The DGN exhibits very desirable properties. First, after the initial trained patterns have spread throughout the memory space, additional patterns can still be stored in the DGN. Secondly, it is possible to distinguish between trained and spread patterns. And finally a trained pattern and its associated spread patterns can be removed without affecting the rest of the stored patterns.

1. INTRODUCTION

The basic component of a weightless neural system [2] is the Random Access Memory (RAM). The inputs to a RAM node consist of N lines whose Boolean values form addresses to which correspond memory locations. The contents of a memory location is seen as the *internal state* in which the location can be. A memory location can be in one of two states, 1 or 0.

When a RAM learns a new pattern, during a *write* operation, the state transition of a memory location can be expressed as:

$$S' = S \cdot (\bar{w} + \bar{a} \cdot w) + a \cdot w [x \cdot 1 + \bar{x} \cdot 0], \quad (1)$$

where S and S' are the states of a memory location before and after the *write* operation, respectively; w is a Boolean variable whose value is 1 during a write operation and 0 otherwise; x represents the Boolean data-in value; and a is a Boolean variable whose value is 1 if the memory location is addressed and 0 otherwise. Memory locations not addressed are not affected by the write operation. This will no longer be the case for the DGN in which the state of a memory location is dynamically updated through interactions with neighbouring locations.

During a *read* operation, the RAM outputs 0 if a 0 state is addressed and 1 if a 1 state is addressed. We write $[0] = 0$ and $[1] = 1$, with $[X]$ representing the value output by the RAM when the memory location addressed is in state X .

Similarly, in a Probabilistic Logic Node (PLN)[3], a memory location can be in one of the states 0, 1 or U. When these states are addressed, during the *use* phase, the corresponding output values are $[0] = 0$, $[1] = 1$ and $[U] = 0$ or 1 chosen randomly with equal probability. For the *train* phase, the equation expressing the state transitions takes a form similar to that of (1):

$$S' = S \cdot (\bar{w} + \bar{a} \cdot w) + a \cdot w [x(s_0 \cdot 1 + s_0 \cdot U) + \bar{x}(s_1 \cdot 0 + s_1 \cdot U)],$$

with $s_X = (S = X)$ and $\bar{s}_X = (S \neq X).$ (2)

If the state of an addressed location is 1 and the data-in value is 0 then the state of this location is reset to U. State 1 is said to be *disrupted*, or to be in *contradiction* with state 0 [4]. Similarly, if the state of the addressed location is 0 and the data-in value is 1, then the state of this location is reset to U.

In a Generalising Random Access Memory (GRAM) [1] the *train* and *use* phases are

identical to those in a PLN. There is an additional phase, called *spreading* [2], during which memory locations not addressed during the training phase, are affected by the use of a *diffusion algorithm* [5][2]. The *spreading* phase only affects memory locations which are in state U.

In order to characterise the influence on a memory location, of its neighbours, a variable g is introduced, which takes one of the values 0, 1, or U: 0, if at least one of the neighbouring locations is in the state 0 and the others are in states 0 or U; 1, if at least one of the neighbouring locations is in the state 1 and the others are in states 1 or U; U, otherwise. For the state transition of a memory location during the spreading phase, using the notation (2) and with $g_j = (g = j)$, we write:

$$S' = \bar{\sigma} \cdot S + \sigma [1 \cdot (s_1 + g_1 \cdot s_U) + 0 \cdot (s_0 + g_0 \cdot s_U) + U \cdot g_U \cdot s_U], \quad (3)$$

with σ being a Boolean variable that takes the value 1 during spreading and the value 0 at other times. (3) is applied synchronously to all memory locations and is repeated a number Γ of times corresponding to a desired *degree of generalisation*.

It is interesting to notice that the GRAM does not have some properties that might be desirable: all the training patterns have to be stored first, prior to any spreading; moreover, a *trained* pattern cannot be distinguished from a *spread* pattern; therefore it is not possible to remove a trained pattern together with its associated spread patterns.

The following section introduces a new weightless node called Dynamically Generalising Neuron (DGN) which operates in two phases. During the first phase, called *learning*, at each time step, each memory location in the DGN updates its state according to its own state, the states of its neighbours and external signals. This dynamical process enables trained patterns to spread automatically throughout the DGN memory space. The second phase is the use phase.

2. THE DYNAMICALLY GENERALISING NEURON

2.1. The states of a memory location

In order to learn additional patterns, after previous patterns have spread their states throughout the DGN, it is necessary to be able to distinguish between states associated with different *spreading levels*. The state of a memory location is said to be at a spreading level i when the memory location is at a Hamming distance i from the memory location which determined its state through spreading. Therefore the state of a memory location will be labelled by adding a suffix i , whose value is the spreading level of the state. A memory location can be in one of the states U, 1_i , 0_i , U_i or R_i , with $i = 0, \dots, N$. U is the initial state of a memory location; its associated output is $[U] = 0/1$. The 1_i are states resulting from the spreading from an addressed memory location trained with a 1 pattern; their associated output value is $[1_i] = 1$. The 0_i are states resulting from the spreading from an addressed memory location trained with a 0 pattern; their associated output value is $[0_i] = 0$. The U_i are states resulting from a contradiction between states 1_i and 0_i ; their associated output value is $[U_i] = 0/1$. The R_i are states assigned to memory locations from which a pattern is being removed; their associated output value is $[R_i] = 0/1$.

2.2. Write and remove operations

These operations affect an addressed memory location ($a = 1$) when the external *write* signal w has the value 1. There is an external *remove* signal r , whose value determines whether the operation is write ($r = 0$) or remove ($r = 1$). An addressed memory location in which a pattern 0 or 1 is written is set to state 0_0 or state 1_0 , respectively. An addressed memory location from which a pattern 0 or 1 is removed is set to state R_0 . The next state S'_{ax} of the memory location, resulting from the interaction with the external signals a , x , w and r can then be written:

$$S'_{ax} = \bar{r} \cdot x (s_{a_0} \cdot 1_0 + s_{a_0} \cdot U_0) + \bar{r} \cdot \bar{x} (s_{1_0} \cdot 0_0 + s_{1_0} \cdot U_0) + r \cdot R_0, \quad (4)$$

$$\text{with } s_{x_i} = (S = X_i) \quad \text{and} \quad \bar{s}_{x_i} = (S \neq X_i). \quad (5)$$

When a pattern is stored at, or removed from a location, by external addressing, the dynamical system formed by all the memory locations of the DGN is no longer in a stable state. It is the interactions between neighbouring memory locations that enable the system to settle in a new stable state for which the node has the correct generalisation with respect to all the trained patterns.

2.3. Interactions with neighbouring memory locations

The influence of neighbouring memory locations (at Hamming distance 1) is taken into account by a variable g , associated with the considered memory location. First, are considered all the memory locations at distance 1 from the location to be updated, whose states have the lowest spreading level. The value of this spreading level is defined as $k - 1$. The states of the chosen memory locations form a set denoted M_{k-1} . The initial state U is defined as having the highest spreading level, say $N + 1$. g takes one of the values $0_k, 1_k, R_k$ or U_k : 0_k , if at least one element of M_{k-1} has the value 0_{k-1} and the others have values 0_{k-1} or U_{k-1} ; 1_k , if at least one element of M_{k-1} has the value 1_{k-1} and the others have values 1_{k-1} or U_{k-1} ; R_k , if at least one element of M_{k-1} has the value R_{k-1} ; and U_k otherwise.

Let the considered memory location be in a state of spreading level i and the value of g characterising the influence of the neighbours be of level k . The next state S'_{dyn} of the memory location, resulting from the dynamical interactions between neighbouring locations, can then be written:

$$\begin{aligned} S'_{dyn} = & s_{R_i} \cdot U_{N+1} \\ & + (i < k) \cdot \overline{s_{R_i}} \cdot S + (i > k) \cdot [s_{R_i} \cdot (G_{0_i} + G_{1_i} + G_{U_i}) + \overline{s_{U_{N+1}}} \cdot G_{R_i} + g_{R_i} \cdot S_{U_{N+1}}] \\ & + (i = k) \cdot [(g_{0_i} + g_{U_i}) \cdot S_{0_i} + (g_{1_i} + g_{U_i}) \cdot S_{1_i} + g_{U_i} \cdot S_{U_i} + (g_{0_i} \cdot s_{1_i} + g_{1_i} \cdot s_{0_i}) \cdot U_i \\ & \quad + s_{U_i} \cdot (G_{0_i} + G_{1_i}) + (s_{1_i} + s_{0_i} + s_{U_i}) \cdot G_{R_i}], \end{aligned} \quad (6)$$

$$\text{with } g_{X_j} = (g = X_j), \quad S_{X_j} = s_{X_j} \cdot X_j \quad \text{and} \quad G_{X_j} = g_{X_j} \cdot X_j. \quad (7)$$

2.4. General update equation

The general update equation for a memory location in a DGN is obtained by combining the effects of write/remove operations (Eq. (4) and (5)) with those resulting from the interactions between neighbouring locations (Eq. (6) and (7)). We write:

$$S' = (\bar{w} + \bar{a} \cdot w) \cdot S'_{dyn} + a \cdot w \cdot S'_{ext}. \quad (8)$$

2.5. Example

Table 1 shows the evolution through time of the contents of characteristic memory locations of a 5-input DGN during the storage of 2 patterns, followed by the removal of one of the stored patterns. The contents of memory locations with addresses 10000, 00000, 01000, 01100, 01110 and 01111 is shown.

3. CONCLUSIONS

In this paper a fundamental viewpoint was adopted in the description of weightless neural nodes, that a memory location contents is no longer considered as the value that is output when a memory location is addressed, but as an *internal state* in which a memory location can be. This has led to the definition of the DGN weightless node.

The DGN was derived from the GRAM, in which the training and spreading operations are performed in a single phase. The DGN is able to store patterns and spread them, via a dynamical process, governed by equations (4) to (8) above, involving interactions between each memory location and its neighbours and external signals. The DGN possesses certain advantages over the GRAM. First, it is possible to store

Time	10000	00000	01000	01100	01110	01111		
$T_1 - 1$	U	U	U	U	U	U	stable	
T_1	0_0	U	U	U	U	U	unstable	
$T_1 + 1$	0_0	0_1	U	U	U	U		
$T_1 + 2$	0_0	0_1	0_2	U	U	U		
$T_1 + 3$	0_0	0_1	0_2	0_3	U	U		
$T_1 + 4$	0_0	0_1	0_2	0_3	0_4	U		
$T_1 + 5$	0_0	0_1	0_2	0_3	0_4	0_5	stable	
...			PATTERN 1					
$T_2 - 1$	0_0	0_1	0_2	0_3	0_4	0_5	unstable	
T_2	0_0	0_1	0_2	1_0	0_4	0_5		
$T_2 + 1$	0_0	0_1	1_1	1_0	1_1	0_5	stable	
$T_2 + 2$	0_0	0_1	1_1	1_0	1_1	1_2		
...	...	...	...	...	...	...		
$T_R - 1$	0_0	0_1	1_1	1_0	1_1	1_2	unstable	
T_R	R_0	0_1	1_1	1_0	1_1	1_2		
$T_R + 1$	U	R_1	1_1	1_0	1_1	1_2	stable	
$T_R + 2$	U	U	1_1	1_0	1_1	1_2		
$T_R + 3$	U	1_2	1_1	1_0	1_1	1_2	stable	
$T_R + 4$	1_3	1_2	1_1	1_0	1_1	1_2		
...			PATTERN 2					

Table 1: Evolution through time of the contents of characteristic memory locations of a 5-input DGN. First two patterns are stored, followed by the removal of one of the stored patterns.

additional patterns even after the spreading of previous patterns. Secondly, it is possible to distinguish between trained and spread patterns. And finally it is possible to remove trained patterns and their associated spread patterns without affecting the remaining stored patterns.

ACKNOWLEDGEMENTS

The author thanks C. Ioannou for useful comments.

REFERENCES

- 1 Aleksander, I., "Ideal Neurons for Neural Computers", *Proc. Int. Conf. on Parallel Processing in Neural Systems and Computers*, Düsseldorf, Springer Verlag, 1990.
- 2 Aleksander, I., "Weightless Neural Tools: Towards Cognitive Macrostructures", *The CAIP Neural Network Workshop*, Rutgers University, New Jersey, October 1990,
- 3 Kan, W. K., and Aleksander, I., "A Probabilistic Logic Neuron Network for Associative Learning", in *Neural Computing Architectures*, Ed. I. Aleksander, Kogan Page, London, MIT Press, Boston, 1989.
- 4 Ntounoufis, P., "Storage Capacity and Retrieval Properties of an Auto-associative General Neural Unit", *Proc. IJCNN-91*, Seattle, 1991.
- 5 Wong, K. Y. M., and Sherrington, D., "Theory of Associative Memory in Randomly Connected Boolean Neural Networks", *J. Phys. A: Math. Gen.*, 22, pp. 2233-63, 1989.