Stochastic arrays and learning networks

Richard A. Leaver

Stochastic Arrays and Learning Networks

Two Volumes – Volume One

Richard A. Leaver

B.Sc. (Dunelm), M.I.E.E.E, A.M.I.E.E.

School of Engineering and Applied Science

University of Durham

August 1988

A Thesis submitted for the degree of

Doctor of Philosophy of the University of Durham.

Abstract

Stochastic Arrays and Learning Networks

R.A.Leaver

This thesis presents a study of stochastic arrays and learning networks. These arrays will be shown to consist of simple elements utilising probabilistic coding techniques which may interact with a random and noisy environment to produce useful results. Such networks have generated considerable interest since it is possible to design large parallel self-organising arrays of these elements which are trained by example rather than explicit instruction. Once the learning process has been completed, they then have the potential ability to form generalisations, perform global optimisation of traditionally difficult problems such as routing and incorporate an associative memory capability which can enable such tasks as image recognition and reconstruction to be performed, even when given a partial or noisy view of the target. Since the method of operation of such elements is thought to emulate the basic properties of the neurons of the brain, these arrays have been termed neural networks.

The research demonstrates the use of stochastic elements for digital signal processing by presenting a novel systolic array, utilising a simple, replicated cell structure, which is shown to perform the operations of Cyclic Correlation and the Discrete Fourier Transform on inherently random and noisy probabilistic single bit inputs. This work is then extended into the field of stochastic learning automata and to neural networks by examining the Associative Reward-Punish (A_{R-P}) pattern recognising learning automaton.

The thesis concludes that all the networks described may potentially be generalised to simple variations of one standard probabilistic element utilising stochastic coding, whose properties resemble those of biological neurons. A novel study is presented which describes how a powerful deterministic algorithm, previously considered to be biologically unviable due to its nature, may be represented in this way. It is expected that combinations of these methods may lead to a series of useful hybrid techniques for training networks. The nature of the element generalisation is particularly important as it reveals the potential for encoding successful algorithms in cheap, simple hardware with single bit interconnections. No claim is made that the particular algorithms described are those actually utilised by the brain, only to demonstrate that those properties observed of biological neurons are capable of endowing collective computational ability and that actual biological algorithms may perhaps then become apparent when viewed in this light.

"If the human brain were so simple that we could understand it, we would be so simple that we could not".

G.E.Pugh

The Biological Origin of Human Values, 1977

Acknowledgements

I would like to express my thanks to my supervisor at Durham, Professor Phil Mars for his expert guidance, encouragement and enthusiasm throughout this project. This research was carried out under a Science and Engineering Research Council (SERC) Industrial Studentship and fully supported for three years by my employers, British Aerospace PLC to whom I am deeply grateful for their trust and commitment.

My visit to the Computer Science Department, Carnegie-Mellon University (CMU), Pittsburgh, USA, was funded by the Durham University Travel Fund and the SERC and I would like to thank Professor H.T.Kung and his research group at CMU for many enjoyable and stimulating experiences and discussions. I should also like to thank Dr David Bounds and Dr Jeremy Ward of the Royal Signals and Radar Establishment (RSRE), Malvern, for their help and encouragement. I am especially grateful to Dr Mansoor Sahardi for early guidance, Dr Peter Green of the Mathematics Department for advice on aspects of the statistical work and Miss Kay Cummins for her skill in preparing the illustrations.

Finally, none of this work would have been possible without the endless patience of Mr Brian Lander and the Durham University Computer Centre operators and advisers in the face of exponentially increasing requests for computer time together with the processing of vast amounts of arcane graphics output.

Declaration

I hereby declare that this thesis is a record of work undertaken by myself, that it has not been the subject of any previous application for a degree, and that all sources of information have been duly acknowledged.

In the course of this research, the following were included in an approved programme of advanced studies:

1. Advanced lectures and examination in the subjects of Digital Systems, Digital Signal Processing and Very Large Scale Integration (VLSI) Specifications and Architectures, January - July 1986.
2. A working visit to the Computer Science Department, Carnegie-Mellon University, Pittsburgh, USA, July - September 1986.

Richard Leaver, August 1988

Copyright

for Kenneth and Sylvia

Volume One

Table of Contents

Volume Two

List of Figures

Chapter 2

Chapter 3

Chapter 4

Chapter 5

Chapter 6

Chapter 7

Chapter 8

Chapter 9

Appendix 2

Appendix 3

Appendix 4

Appendix 5

List of Tables

Chapter 3

Chapter 5

Chapter 6

Chapter 7

Chapter 8

Appendix 5

Appendix 6

Chapter One

Introduction

1.1 Objectives

This thesis presents an examination of arrays of simple neuron-like elements which utilise stochastic coding techniques. These arrays may interact with a random or noisy environment and through essentially local operations and decisions, exhibit a collective computational ability which offers considerable scope for novel processing of problems which require that many inputs are examined simultaneously and an optimum or near-optimum decision generated. Current research in this field is generating much interest and has been prompted by studies from both biology and physics with particular emphasis on the unique cognitive properties of the human brain, [1],[2],[3],[4],[5],[6],[7],[8].

1.2 The Human Brain

The human brain is a parallel information processing machine of incredible power, easily capable of outperforming todays serial or limited-parallel computers in many areas involving decision or discrimination. Typical areas in which it excels are the seemingly effortless identification of images from noisy inputs, self learning from observation and experience, without any external programming of internal states, and the ability to deal with new

situations using inference from previous experience. This is more remarkable, when one considers that the individual processing element or neuron, is a simple organic cell operating at speeds measured in milliseconds. Since human picture recognition is perceived as occurring with intervals of the order of tenths of a second, a limit of approximately one hundred serial steps has been suggested for neural operations of this kind. Additionally, each simple neuron may possess over ten thousand connections of variable connectivity to other neurons, [9], [10]. This suggests that the the brain is a massively parallel, serially shallow network of simple elements with information encoded via connection strengths between neurons and distributed across the neural network.

1.3 Neural Networks

Such properties have two main advantages to the neural network. The first is that local damage to the network does not necessarily result in a significant loss of information. The second advantage is that this method of encoding information represents an efficient means of performing Associative Memory, whereby presentation of a few key features enables recall of a wealth of information connected with those items. Of particular interest is the facility to reconstruct the most likely image from a noisy or partial view, especially when the surrounding environmental context and previous experience is used to provide additional information.

If these features could be encapsulated into assemblies of highly interconnected artificial neurons, then such networks could begin to offer low

level primitive operations which might then be built into higher level symbolic structures similar to those which predominated in Artificial Intelligence research in recent years. Whereas organic neurons operate at speeds measured in milliseconds, their electronic counterparts could operate in nanoseconds and therefore efficient self-organising electronic neural networks have the potential for incredible analysis and recognition performance. Such operation is likely to operate in conjunction with a standard host computer. The learning mechanisms yield inherent fault tolerance. Additionally, since each element takes part in the global decision, no silicon would remain idle in a Very Large Scale Integration (VLSI) implementation. These features could lead to successful Wafer-Scale Implementation (WSI).

1.4 Overview of the thesis

The thesis is arranged for the convenience of the reader in two volumes. The first volume describes the research and results obtained together with explanatory appendices. The second volume contains the key computer program listings which enables reproduction of any of the simulations reported and are documented in the appendices of Volume One.

The thesis commences by introducing the concept of stochastic encoding and explaining how useful mathematical functions may be computed using these probabilistic pulse sequences, [11]. An original design for a stochastic systolic array is given as an example of the inherent simplicity and noise tolerance of this approach, [12]. The theory of stochastic learning automata is then reviewed. Such automata have proved highly successful for adaptive

routing in communication networks, [13], [14]. The thesis then describes connectionist or neural networks. The use of stochastic learning automata in such networks is described together with a novel stochastic implementation of a deterministic connectionist algorithm. Original learning rate plots are presented for different networks performing non-trivial tasks, using a series of comparative performance evaluations.

As future work, the thesis considers how all the previous approaches may be considered to be simply variations on a single key stochastic element or neuron which may be biologically plausible and suggests some hybrid approaches. Each chapter is briefly summarised below.

1.4.1 Stochastic Computation

Chapter Two reviews the theory of stochastic sequences and describes how they enable amplitude information to be conveyed using a frequency coding method, where the probability of an event occurring is used as the means of communicating an estimate of the amplitude over a single bit line, [15]. Such stochastic sequences are more familiarly known as Bernoulli sequences, [16]. The theory of Bernoulli sequences is examined, enabling the statistical errors associated with these methods to be derived. Although the data bandwidth is considerably reduced, compared to standard binary representations due to the need to take a time average, the use of probability for encoding is shown to reduce basic computation operations to simple logic elements.

1.4.2 Stochastic Systolic Array

Chapter Three shows how simple stochastic elements may interact to compute useful mathematical functions such as those necessary for digital signal processing, [17]. As an example, a novel stochastic systolic array design is used to evaluate the computational properties of probabilistic coding. It is described how the array can compute the Cyclic Correlation function and thereby, for odd prime transform lengths also determine the Discrete Fourier Transform, [18]. A series of simulations are presented to demonstrate the properties of the array.

Since the array possesses local, synchronous, single bit connections, it may operate at high speeds. The probabalistic nature of the array could enable direct input of stochastic quantities such as the output of photomultipliers for photon correlation as used in applications such as laser doppler anemometry, [12], [19]. The high degree of noise tolerance inherent in stochastic encoding is demonstrated and it is described how the total hardware involved is trivial and potentially fault tolerant offering a low precision, fast estimation facility.

These results are encouraging in that they suggest that the numerical properties of the stochastic array are, in a simple sense similar to those characteristic of the brain. Whilst meaningful comparisons of the stochastic array with deterministic alternatives are difficult, the chapter concludes with a discussion outlining the advantages and disadvantages of this approach over more conventional signal processing.

1.4.3 Stochastic Learning Automata

Chapter Four reviews stochastic learning automata which seek to maximise a reinforcement signal from some external agent, or environment, in order to respond optimally, [20]. It is explained how these automata essentially update on each trial, an internal table, which holds the probabilities of selecting particular actions. Learning automata have been found to exhibit close to optimal performance for communications network routing applications, where nodes have a high failure rate, [13], [14]. The simple stochastic hardware associated with these elements, combined with essentially local computation may be shown to enable global optimisation of particular network characteristics and this has attracted attention for their use in neural networks.

1.4.4 Neural Networks

Chapter Five reviews neural networks by presenting a simple overview of a biological neuron and discussing the key aspects which might be incorporated into a neural model. The extension of the model to a network capable of collective computational behaviour is then described. The necessity of requiring 'hidden elements', not directly connected to the problem, but nevertheless influencing the behaviour of the network so that it can solve 'interesting' problems is explained. The chapter then describes how the central key problem at the heart of all neural network research is the 'Credit Assignment' problem whereby prospective network training algorithms require a method for efficiently adapting the connection strengths or 'weights' in these hidden elements so that the network operates correctly.

The various approaches to this problem utilised by the different neural network algorithms and architectures are described. This chapter concludes the first part of the thesis. The second half brings together stochastic encoding techniques and learning automata with neural networks.

1.4.5 Associative Reward–Punish Learning Automata

Chapter Six introduces the Associative Reward–Punish (A_{R-P}) stochastic learning automaton which can perform pattern classification, [21], [22]. It is shown how this element can form a particular class of neural network utilising reinforcement learning methods. Simulations are presented which evaluate the behaviour of the A_{R-P} element both for simple and for non-trivial problems. The limitations of the basic A_{R-P} element are discussed and novel methods of training arrays of these elements are presented together with further results.

1.4.6 Stochastic Implementation of Neural Networks

Chapter Seven describes a possible hardware implementation of a stochastic neural element similar in operation to the Inner Product Cell described in Chapter Two. The chapter then considers the extension of the neural implementation to a novel stochastic A_{R-P} element. Original simulations are presented for this case following the examples given in Chapter Six. The method is then extended to an original stochastic implementation of the error backpropagation deterministic neural network described in Chapter Five. Simulations are presented for both the deterministic and stochastic

error backpropagation elements.

1.4.7 Comparative Performance Evaluations

Chapter Eight presents a series of non-trivial problems which demonstrate the strengths and weaknesses of each of the different learning methods summarised in the previous chapters. Comparative simulations are performed using parameters studied in the previous chapters and relative performance histograms are presented which enable a critical assessment to be made.

1.4.8 Further Work

Chapter Nine concludes the thesis with a discussion of the work presented and considers future work involving studies of novel hybrid schemes based on a fundamental stochastic element.

1.5 Conclusions and Summary – Chapter One

This chapter has presented a brief introduction and overview of the thesis. It is shown that the nature of stochastic processing, with its blend of both analogue and digital computation is a natural medium for neural network operation and that all such stochastic elements described, systolic, automata or neuron may be considered facets of an underlying simple element, which, in a highly parallel array may exhibit a collective computational response, yielding a global solution from essentially individual local operations.

Chapter Two

Stochastic Computation

2.1 Introduction

This chapter introduces the concept of stochastic as opposed to deterministic computing. Stochastic computing may be considered to be a digital extension of analogue computing, where analogue voltages are replaced by random pulse trains and analog integrators are replaced by binary counter mechanisms, [11], [15], [23], [24], [25], [26], [27]. In the later chapter reviewing neural networks, it will be described how analogue devices have been used to implement various features exhibited by the neurons of the brain in actual hardware. The application of stochastic processing to neural networks therefore forms a logical extension of current research.

This chapter will explain the principles and limitations of such processing in order to provide a basic theoretical framework for later chapters. Although many published methods of training neural networks do in fact utilise a probabalistic approach, it is believed that none have directly considered the principles of stochastic computation as described in this chapter (although it has been previously mentioned in the stochastic literature, [28]). By examining these algorithms from this viewpoint, some degree of unification may be made.

2.2 Stochastic Encoding

Stochastic encoding involves representing a positive normalised value by a probability expressed as a digital bit serial pulse stream using the Binomial or Bernoulli distribution, [16], [29]. This is performed by thresholding the input value against a random level between 0 and 1. If the random level falls below the value, then a pulse is generated, otherwise not. The random level is usually obtained using standard pseudo-random number generators or from a noise source.

If the normalised value to be represented is p, then the probability of a pulse occurring is P(ON) = p in a single trial. Therefore q = 1-p is the probability that, in a single trial a pulse does not occur, ie. P(OFF). A necessary condition is that the trials must be independent, ie. they do not influence each other. In an infinite sequence, the mean or time average is equal to the generating probability p. In a finite sequence, the time average forms an estimate of p. An example of a stochastic sequence approximately representing 0.4 is depicted in Figure 2.1.

2.2.1 Mappings

Probability is expressed as a value between the limits 0 and 1. Consequently the values which are to be represented must be mapped in between these limits. This introduces a fundamental problem in that a logarithmic representation as used by deterministic computers may no longer be used since the mapping must be linear. Therefore the range of representation of values is limited and stochastic methods cannot manipulate values with

any great accuracy.

Although there are a number of mappings which may be employed to encompass both positive and negative values, the method employed by the networks described in this thesis is to utilise two separate lines. The first (p) line represents positive values and the second (n) line represents negative values. This representation is known as the Dual Line Symmetric Representation (DLSR). For clarity, stochastic components will be shown as if referring to a single positive line unless otherwise stated as extension to the DLSR representation is trivial.

2.2.2 Accuracy of representation

As described, stochastic encoding involves the use of Bernoulli sequences to convey information. It is convenient to describe the behaviour of a system operating under stochastic principles using the moments of the distribution, [16], [29]. The first moment is the **expected** or **average** (mean) value. An estimate of the generating probability, p is obtained by taking the time average of the pulse sequence. As the length of the sequence tends to infinity, the estimate tends to the actual value. The estimate of the generating probability, $\widehat{P}$ obtained by taking the finite average as described will follow a binomial distribution about the mean value $\bar{P}$. As the interval becomes infinite, the estimate becomes equal to the generating probability.

If one were to take independent estimates of $\widehat{P}$ from independent sequences of the same length, e.g. by taking a moving average at intervals over the sequence (explained later) then the dispersion of each estimate about

the mean of all the estimates may be expressed in terms of the second moment about the mean, the **variance** σ^2. It is convenient to take the positive square root of the variance, ie. the **standard deviation** σ as a measure which bounds the proportion of such estimates lying within the range

$$\bar{P} \pm \sigma \tag{2.1}$$

The Bernoulli distribution is only symmetric about the mean for a generating probability P=0.5. For this mid-range case only, 68% of the estimates taken as described should fall in the range $\bar{P} \pm \sigma$. Appropriate definitions, together with the derivation of the standard deviation of a Bernoulli sequence, σ are given in Appendix One. It is shown that, over N trials, σ is given by;

$$\sigma = \sqrt{\frac{P.(1-P)}{N}} \tag{2.2}$$

It is important to consider the implications of this result. Clearly for P = 0 or 1, ie when the signal is deterministic, σ is zero. Maximum σ occurs for the mid-range case when P = 0.5. Figure 2.2 plots this function for N=100 trials. We may therefore expect that the mid-range precision is roughly 10% accurate over N=100 trials, (in the sense that 68% of the estimates fall in the region bounded by the standard deviation), 1% accurate over N=10,000 trials and 0.1% accurate over N=1,000,000 trials. When considering the use of stochastic encoding, it is necessary to compare the waste of bandwidth against the resulting ease of performing mathematical computations.

2.3 Mathematical Operations

If each sequence is statistically independent, then it is possible to perform mathematical operations using simple logic components. The resulting sequence is then time averaged in order to convert the probabilistic output into deterministic form. It is first necessary to consider how such operations will affect the sequence and whether the resulting sequences remain Bernoulli and this is crucial for successful subsequent operations of the same type.

2.3.1 Scaling and Dynamic Range

Using probability fields to represent numbers requires care in ensuring that the system has adequate dynamic range for the proposed calculation. Probability only has meaning between the values 0 and 1. Clearly then, if one wishes to represent a number in this fashion, it must lie between these limits and so input data must be normalised relative to 1. If it is wished to represent for instance, all numbers between 0 and 10, then all numbers must be normalised by dividing by ten such that the value 1 is represented by 0.1, 2 by 0.2 and so on up to 10 as probability 1. The probability field, ie. denominator, is therefore 10. Clearly the product of two fractions in this field e.g. 0.1 and 0.2 (representing the numbers 1 and 2) will be 0.02, represented by a probability of 2 in 100.

The probability field after multiplication is the product of the probability fields of the multiplicands. This is exactly the same as wordlength growth in digital systems. However in the stochastic system, there is no architectural disadvantage brought by such field growth but the probability

field must take account of the maximum expected value which the system must encompass. Repeated multiplication is to be discouraged as it can be seen that this may result in considerable attenuation of the output probability.

2.3.2 Multiplication

Multiplication in stochastic systems is trivially easy. As an example, consider multiplying 0.1 by 0.2. The probability of a pulse occurring in the formers stochastic representation will be 1 in 10 and that of the latter, 2 in 10. Clearly the probability of the two occurring at the same time is represented by the product of these two probabilities **only** if the two sequences are statistically independent.

This may be accomplished by the use of an AND gate to compare the two input stochastic streams. This is shown in Figure 2.3. If there are coincident pulses then an output pulse is generated for each coincident event. When averaged, the number of output pulses divided by the total number of slots for a suitable interval will tend to 0.02, which is the correct answer. Note that for the DLSR representation, four AND gates are required, one for each combination of positive and negative lines. It is very important that the multiplicands are derived from independent random sequences, otherwise the process fails. Consider the product C of two random sequences A and B, ie.

$$C = A.B \tag{2.3}$$

Then for independent random binary sequences only, an expression for the expectation of a pulse occurring in the product sequence of the two input

sequences A and B is given by;

$$E[C] = P(C) = P(A).P(B) \tag{2.4}$$

Therefore substituting P = P(C) into equation 2.2 for σ:

$$\sigma_c = \sqrt{\frac{P(C).(1 - P(C))}{N}} \tag{2.5}$$

and so

$$\sigma_c = \sqrt{\frac{P(A).P(B).(1 - P(A).P(B))}{N}} \tag{2.6}$$

Other mathematical functions involving multiplication may be obtained as follows [15]. Inversion for the single line case is simply performed using an inverter as shown in Figure 2.4a and for the DLSR representation, by crossing over the positive and negative lines as shown in Figure 2.4b. Since a bit-delayed stochastic sequence should be independent of the non-delayed sequence, squaring is simply performed by delaying the sequence through a latch and ANDing the result with the input sequence as shown in Figure 2.5. The special operation $A(1 - A)$ where A is defined between 0 and 1, may be performed using a bit delay, inversion and AND operation (Figure 2.6). The relevance of this operation is explained in Chapter Seven. Note, operations involving bit-delay with subsequent operations on themselves effectively result in sequences with limited dependence and therefore not strictly describable as a Bernoulli sequence although they may be used as such. Integration is obtained by the use of an Up/Down counter (Figure 2.7).

2.3.3 Addition

Addition is less trivial but still simple. Recall that the value of a number is represented by the number of times a pulse occurs. Clearly then, two values may be added together by using an OR gate (two are required for the DLSR representation) as shown in Figure 2.8. The resulting bit stream represents the summation. Unfortunately there are two problems associated with this. The first is that the method does not take account of coincident pulses, in other words if the probabilities are P(A) and P(B), then use of the OR gate alone gives the result P(A)+P(B)-P(A)P(B) where P(A)P(B) is the probability of coincident pulses. This results in inaccurate addition.

It may be seen that when P(A) and P(B) are small values relative to the total dynamic range, ie. the probability field is large, the product P(A)P(B) becomes small but this is an unsatisfactory solution. The standard method for stochastic addition is to ensure that the coincident case does not arise. This is performed by multiplying the two sequences to be summed by a random sequence of generating probability 0.5 as shown in Figure 2.9. Then with this arrangement there are no coincident pulses. However the resulting sum has been halved, clearly repeated addition will be unsatisfactory and is therefore not used in this research.

The second problem, is that it must be ensured that the summation does not exceed the total probability field available otherwise a limiting overflow occurs. This is solved by scaling the field to cope with the maximum expected value ie. set the system dynamic range appropriately. The problem of coincident pulses may be alleviated by the following various methods as

follows.

2.3.4 Addition Correction Methods

This research has examined several potential solutions to the problem of addition using an OR gate. If one counts the number of coincident pulses occurring and then re–inserts these back into the bit stream after the OR operation, when the output of the OR gate is low, the result may be corrected. This can be accomplished using two up/down counters and logic gates. This is shown as a simple schematic for one line in Figure 2.10 and operates as follows. For each coincident event, the counter is incremented up by one. When the counter has non–zero content then its output is high, otherwise low. The counter output is ANDed with the inverse output of the initial OR addition operation. If the output of the OR operation is LOW, then the counter is decremented by one and a pulse inserted at the output of the adder. If the counter is empty then decrementing is disabled.

The counter is also designed so that if the counter representing the positive line contains pulses and the counter representing the negative line also is in the non–zero state, then these negate each other appropriately. If the system is operating near the maximum of its dynamic range, the counter will be required to count a large number of coincident events as there will be very few occasions when the output of the OR gate is low and pulse re–insertion will be infrequent. This situation may be alleviated by scaling the system dynamic range to be greater than the maximum expected output value. This reduces the number of occasions when the circuit cannot

insert pulses and means that the counter does not need to have large capacity. However increasing the scaling results in adverse probability attenuation which affects the length of the interval necessary to obtain a satisfactory estimate. Therefore a tradeoff is necessary between design and performance.

If the counter contents, on average remain relatively low to the number of trials, then the resulting sequence may still be considered to represent the value of the resulting summation using a sequence which closely follows the Bernoulli distribution. Addition of two random sequences A and B, results in a sequence C whose Standard Deviation σ_c is related to the Standard Deviations σ_a and σ_b as follows. If the sequence remain close to Bernoulli, then since

$$E(C) = P(C) = P(A) + P(B) \tag{2.7}$$

then if there is no coincident pulse case (which would cause error), substituting into equation 2.2 for σ gives;

$$\sigma_c = \sqrt{\frac{P(C).(1 - P(C))}{N}} \tag{2.8}$$

and so

$$\sigma_c = \sqrt{\frac{(P(A) + P(B)).(1 - (P(A) + P(B)))}{N}} \tag{2.9}$$

which may be alternatively expressed using the standard expression for the addition of two random sequences;

$$\sigma_c{}^2 = \sigma_a{}^2 + \sigma_b{}^2 + 2COV(A, B) \tag{2.10}$$

If the sequences are independent, the Covariance term is zero and using equation 2.2, both expressions reduce to;

$$\sigma_c = \sqrt{\sigma_a{}^2 + \sigma_b{}^2} \tag{2.11}$$

Other alternative approaches have been examined. By noting that the error involved in addition using an OR gate is simply the probability P(A)P(B), then by adding this value, again using an OR gate to the output of the adder more accurate addition will take place as long as care is taken to ensure statistical independence and realising that the resulting sequence has some degree of dependence. This is known as one–level cascading and is shown in Figure 2.11. Further cascading may take place, although this gives less benefit for more complexity. Two–level cascading is shown in Figure 2.12.

2.4 Random Number Sequences

Stochastic processing depends on the use of random sequences generating equiprobable values between 0 and 1. A convenient way of performing this is to use a comparator with noise input as shown in Figure 2.13. The noise input may be analogue in origin or from a digital source, using standard methods involving feedback shift registers. It is essential when performing stochastic operations, that the two sequences are independent. Otherwise a correlation effect will take place between the two and an incorrect result obtained. With care, just one random sequence generator may be used, since by using simple delay methods, a particular sequence should be independent of a bit delayed version of itself, [30],[31].

2.5 Output Conversion

The conversion of a stochastic sequence to a deterministic value is simple, [32], [33], [34], [35], [36]. An output counter increments for each pulse

received, this value is divided at convenient intervals by the total number of clock cycles to give an estimate of the deterministic value. If the counter is designed to have a length which is a power of two, then division becomes a simple binary shift operation.

The averaging process is commonly performed using a moving average via a shift register and counter as shown in Figure 2.14. This requires only that the initial 1 bit binary value entering the register and the final value leaving the register are input to the summer rather than continually adding up all the values within the register. This method is adopted for the majority of the processing described in this thesis since it enables 'start-up' values which might otherwise bias the result, to be passed out.

An additional benefit of stochastic processing involves performing the same computation using N processors in parallel. Since each sequence is independent, results may be combined to form a result to greater precision. This is known as bundle processing and is fault tolerant since the output probabilities of each line should each be approximately equal and so an error in one line is easily detectable, [37].

2.6 Special Sequences

If there is an application where the exact operand is known in advance and is fixed, e.g. the precomputed data coefficients for the Discrete Fourier Transform, then only one of the sequences need be derived from a random sequence. For a given time which must be a known total time, or a fixed time block, the known sequence may be formed by just two consecutive states,

one ON, and one OFF, the proportion being dependent on the value being represented.

This is just one potential ordering of a random sequence and may be accomplished by gating triggered from some kind of system clock counter. The product sequence obtained by multiplying this special sequence with a normal stochastic sequence should not be used in further multiplicative stochastic computation since it no longer retains the necessary statistical properties, however it still forms a valid final answer as long as the result is only taken over complete intervals of the time block.

2.7 Inner Product Element

In order to evaluate the properties of stochastic computation, an element has been designed which performs the multiply/accumulate operation using the novel approach of stochastic encoding as shown in Figure 2.15. This element is known as an **inner product** element and can be combined into arrays of identical elements which may compute useful mathematical functions.

This element will form the basis of the systolic array in the next chapter and operates as follows. The inputs to the element at time t are the suitably scaled stochastic sequences, a_t, b_t and c_t. At time t+1, the element computes the function

$$c_{t+1} = a_t.b_t + c_t \tag{2.12}$$

and

$$\begin{aligned} a_{t+1} &= a_t \\ b_{t+1} &= b_t \end{aligned} \tag{2.13}$$

2.7.1 Inner Product Element Simulation Results

The operation of the Inner Product element was investigated by computer simulations using various test parameters. Figure 2.16a shows the static average of the output stochastic sequence c_{t+1} for $t = 1, 10000$ trials where a_t is a stochastic sequence representing 0.5, similarly $b_t = 0.5$ and $c_t = 0.0$. The output range is scaled from 0 to 1. It may be seen from equation 2.9 that the output average should tend to the value 0.25 as indicated by the dotted line. Figure 2.16b plots the difference between actual and expected value. The computation involves only multiplication and no form of addition takes place. The simulation was repeated for the output range scaled between 0 and 2 and these results are shown in Figures 2.17a and 2.17b.

Figure 2.18a shows the average output c_{t+1} where $a_t = 0.5$, $b_t = 1.0$ and $c_t = 0.5$. Note that the expected output for this case is 1.0 which is the maximum probability which may be encompassed with the output range scaled from 0 to 1. The plot shows four methods of addition; counter, one-level cascade, two-level cascade and OR gate. The dotted line represents the theoretical result of 1.0. Each sub-simulation, counter, OR gate etc. was performed using the same random number sequence to allow comparison throughout these simulations. Figure 2.18b plots the difference between actual and expected value. These were repeated in Figures 2.19a and 2.19b for the output range scaled between 0 and 2.

The results clearly show the accuracy of the counter method as compared with the basic OR gate method of addition. The cascade methods

are better than the OR gate and are improved by overscaling the system range which reduces the number of coincident pulses in the OR operation. The counter mechanism was used for future array simulations using this cell in this thesis. This example simulation was taxing in that, when the system was scaled between 0 and 1, the answer was 1.0. In other words the counter was required to reinsert pulses such that all available slots were filled.

The states of the counter were examined to confirm that the average contents remained low. Clearly if the counter was continually incrementing to large values, relative to the number of trials, then the output sequence would be incomplete and the average output would be in error. The counter state is plotted in Figure 2.20a for the scale 0 to 1 simulation, and in Figure 2.20b for the scale 0 to 2 simulation. Note how Figure 2.20a demonstrates that the counter contents are continually increasing although by a small amount. This shows that there are a number of pulses which should have been reinserted to compensate the pulse stream but that no slots were available. The contrast when the system is overscaled is shown by Figure 2.20b when the counter state is rarely greater than two or three pulses held.

Figure 2.21a shows the average output c_{t+1} where $a_t = -0.5$, $b_t = 0.5$ and $c_t = 0.25$ for all addition methods with the output range 0 to 1, and repeated in Figure 2.21b for the output range 0 to 2. The expected average output for this case is 0.0 and represents a good test of the element for this reason. This plot effectively shows the difference between actual and expected value. Note how the performance of each method is nearly identical. This is because of the mechanism which detects the presence of the case of coincident

pulses on the positive and negative lines and negates them before addition is necessary.

2.8 Conclusions and Summary – Chapter Two

This chapter has described the fundamental components of stochastic computation and described how how the operations of multiplication and addition become trivially simple when using sequences coded in this way. The statistical errors involved when using Bernoulli sequences to convey information have been described and simulation results have been presented for a novel simple element which may be replicated into larger arrays to compute useful mathematical functions.

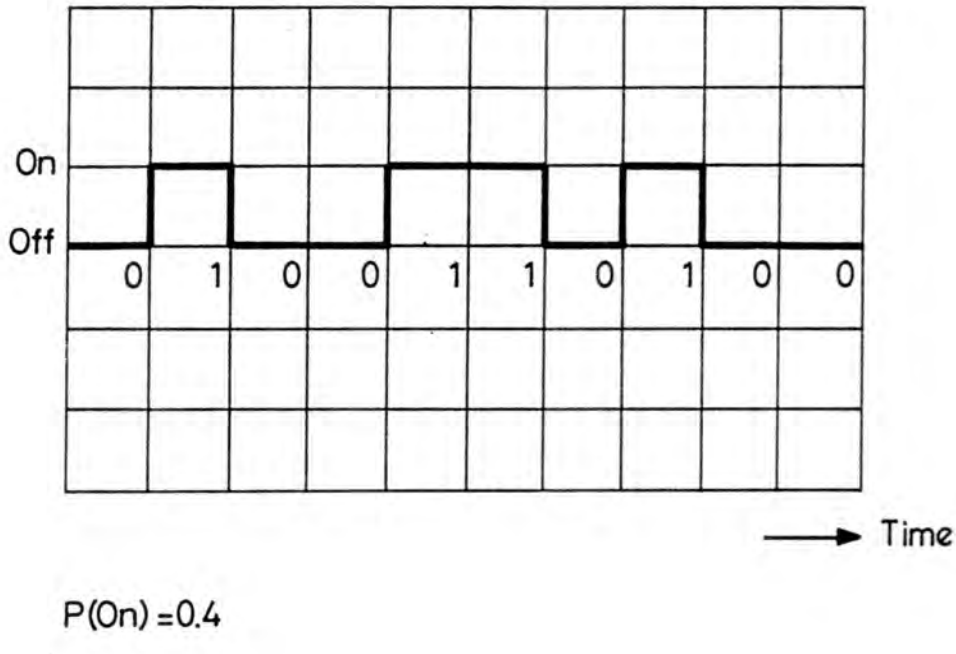


Figure 2.1 Bernoulli Sequence approximately representing the value 0.4

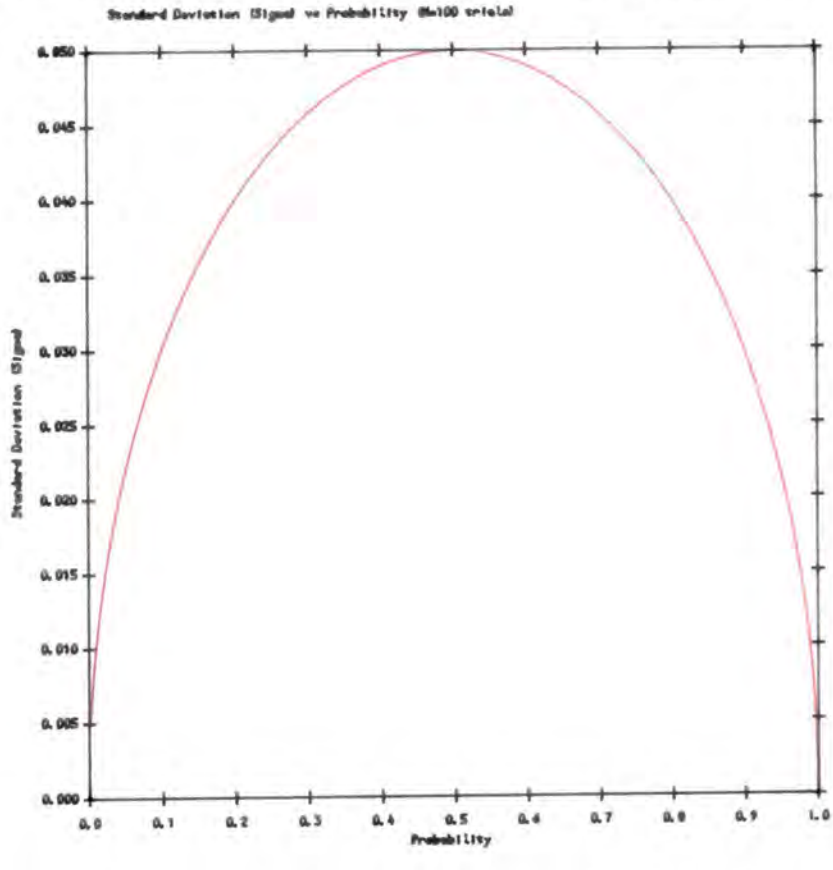


Figure 2.2 Standard Deviation vs Input Probability for a Bernoulli Sequence

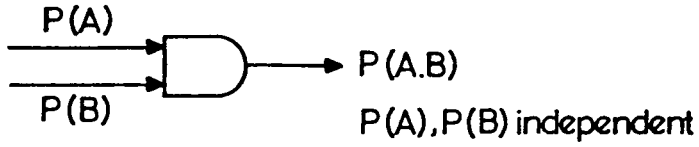


Figure 2.3 Stochastic Multiplication

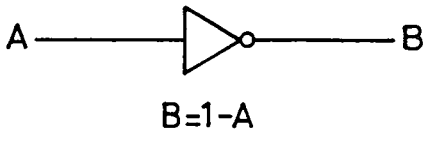


Figure 2.4a Stochastic Inversion - single line

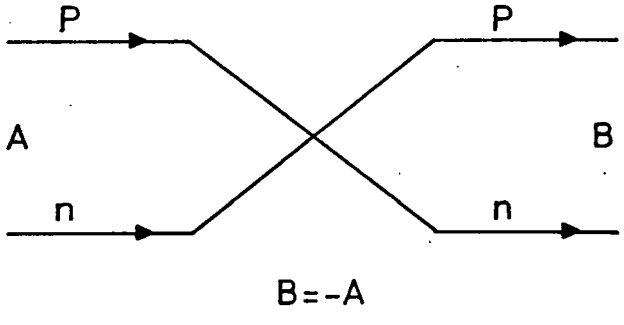


Figure 2.4b Stochastic Inversion - DLSR representation

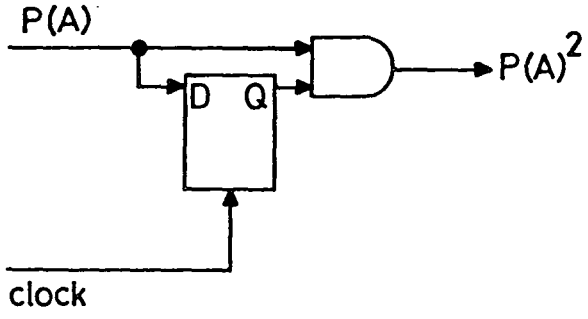


Figure 2.5 Stochastic Squaring

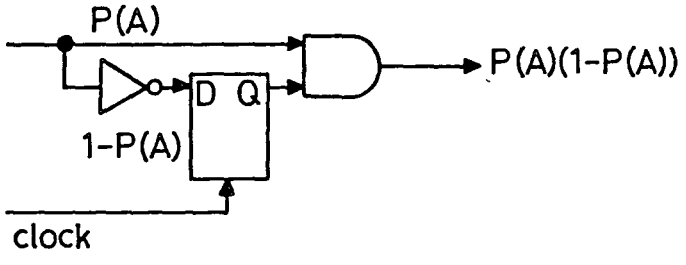


Figure 2.6 Stochastic computation of A(1–A)

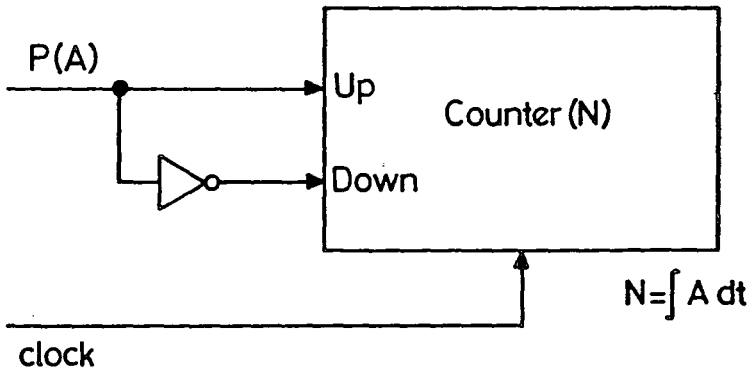


Figure 2.7 Stochastic Integration

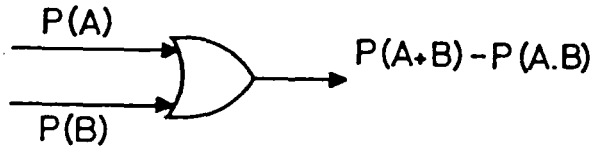


Figure 2.8 Stochastic Summation – OR gate

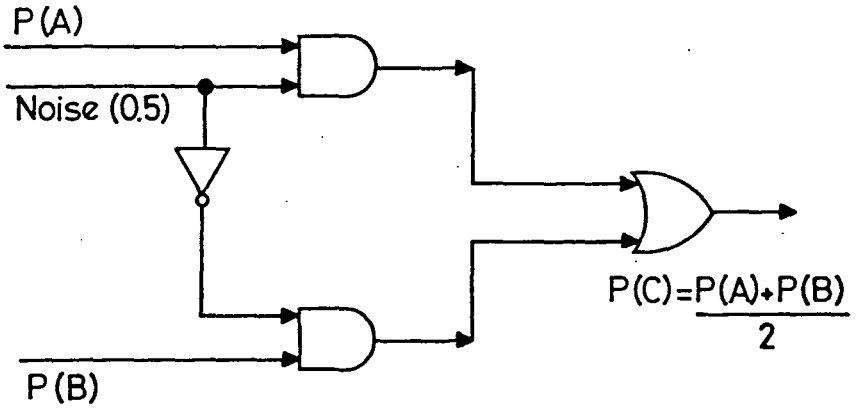


Figure 2.9 Stochastic Summation – random sequence

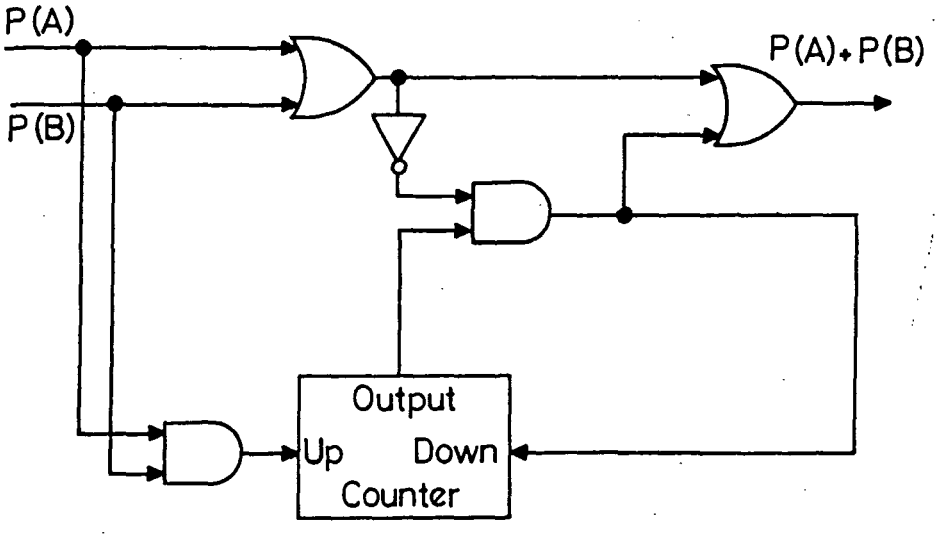


Figure 2.10 Stochastic Summation – Counter

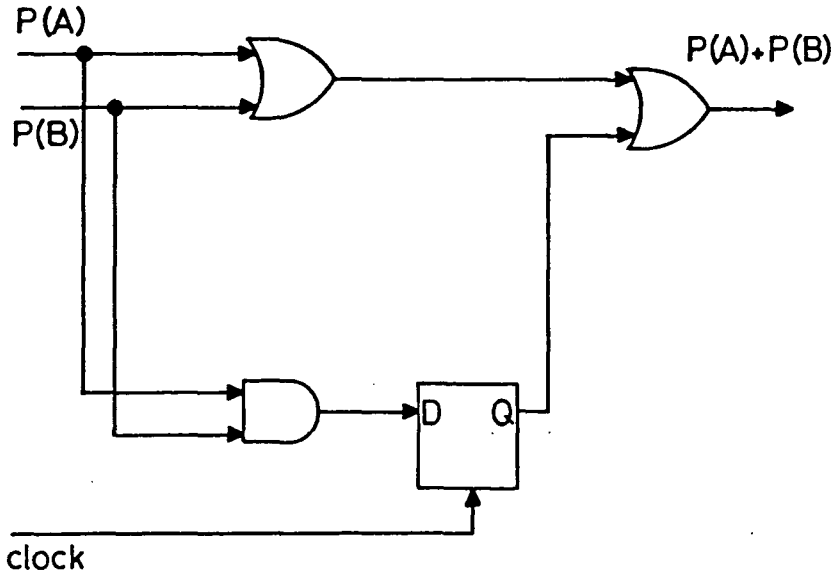


Figure 2.11 Stochastic Summation – one–level cascade

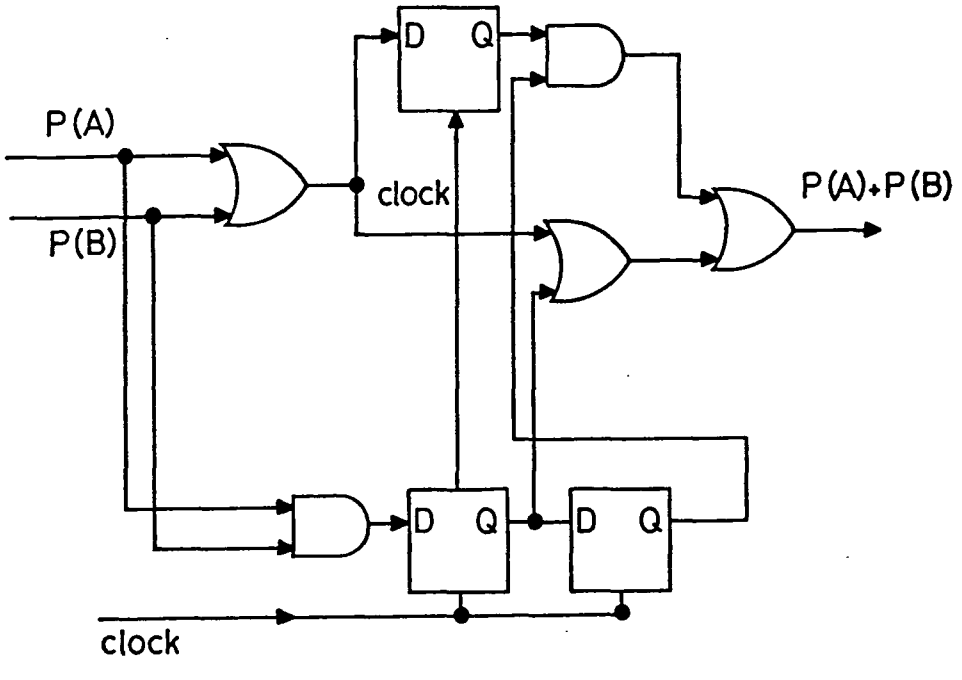


Figure 2.12 Stochastic Summation – two–level cascade

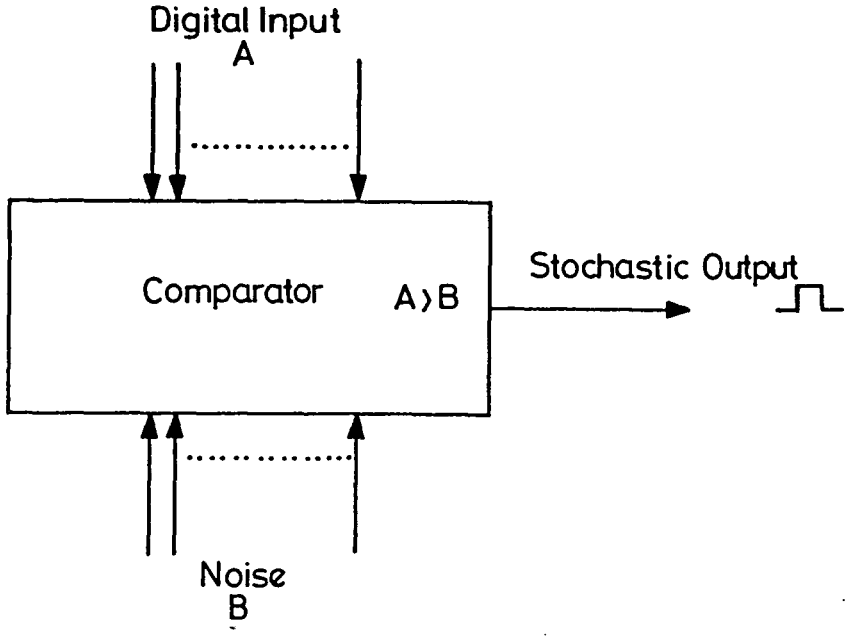


Figure 2.13 Stochastic input sequence generation

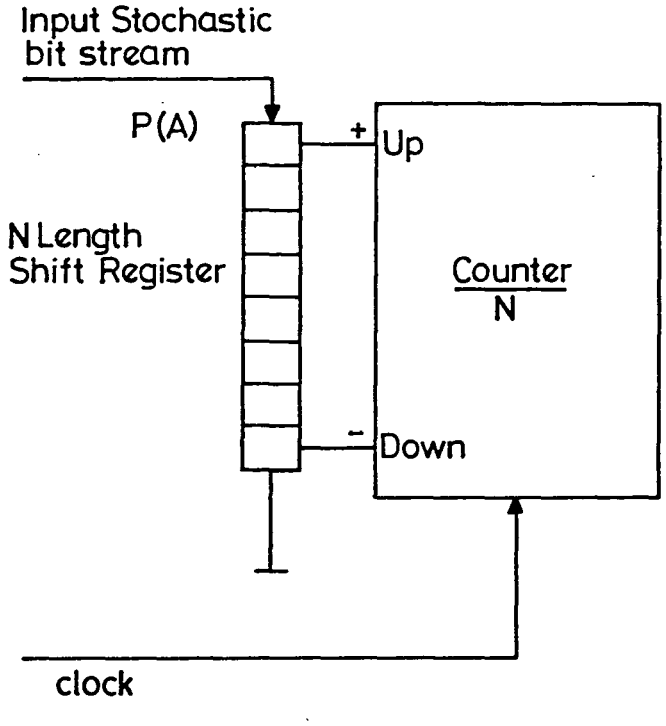


Figure 2.14 Stochastic output sequence conversion

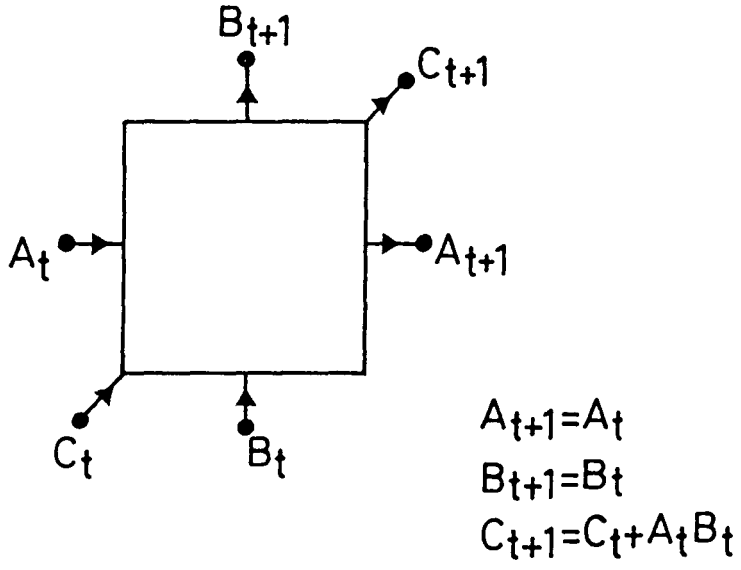


Figure 2.15 Inner Product Element

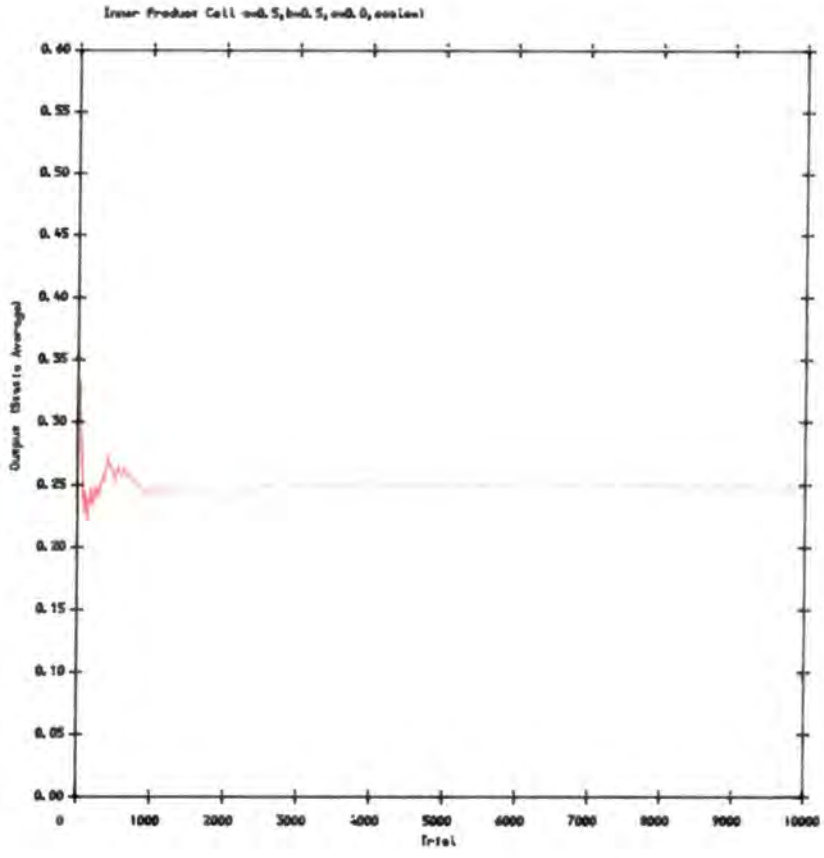


Figure 2.16a Element Output – a=0.5, b=0.5, c=0.0, scale 1

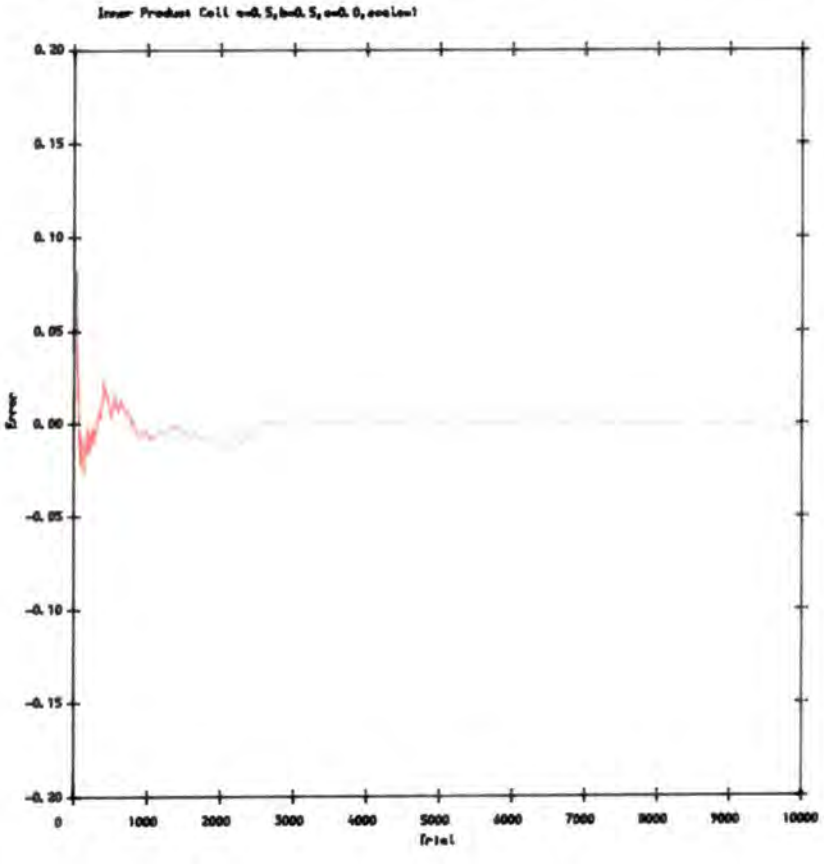


Figure 2.16b Element Error – a=0.5, b=0.5, c=0.0, scale 1

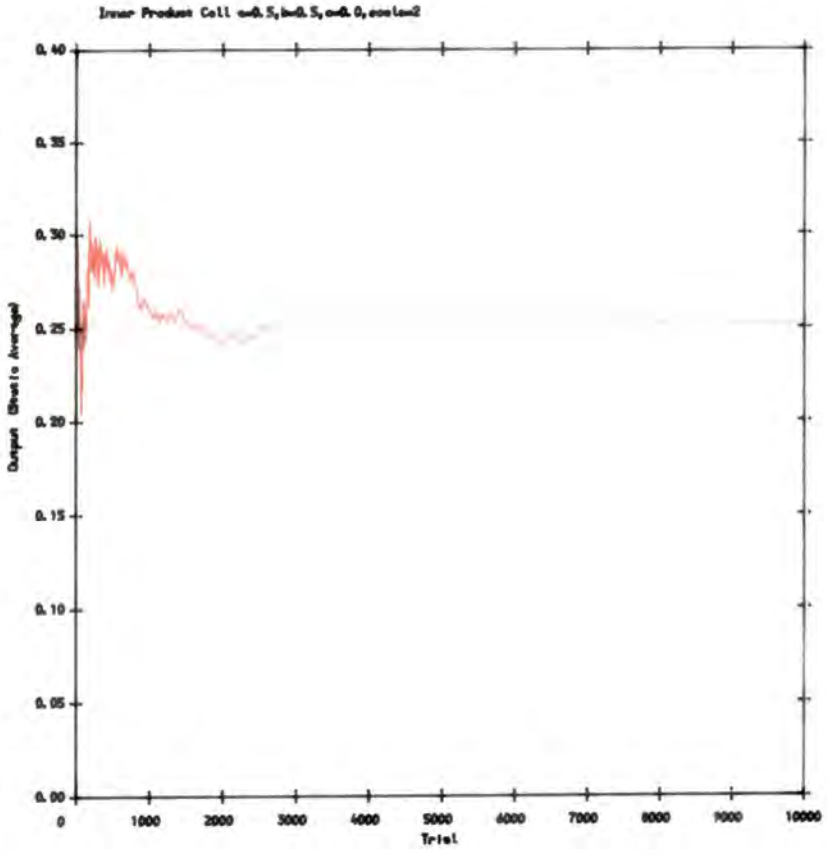


Figure 2.17a Element Output – a=0.5, b=0.5, c=0.0, scale 2

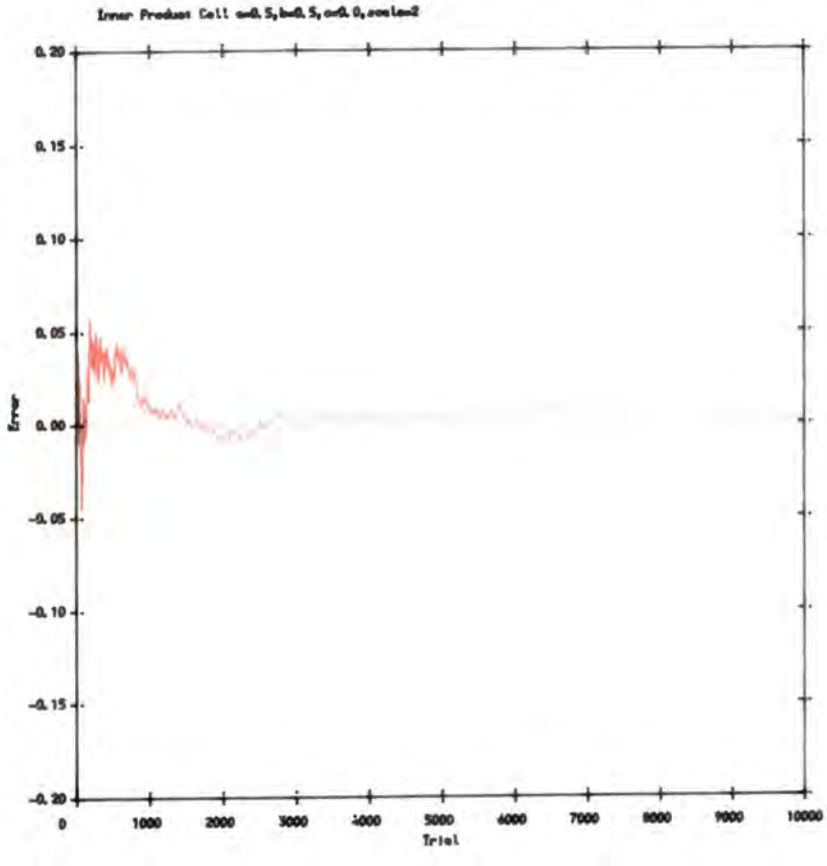


Figure 2.17b Element Error – a=0.5, b=0.5, c=0.0, scale 2

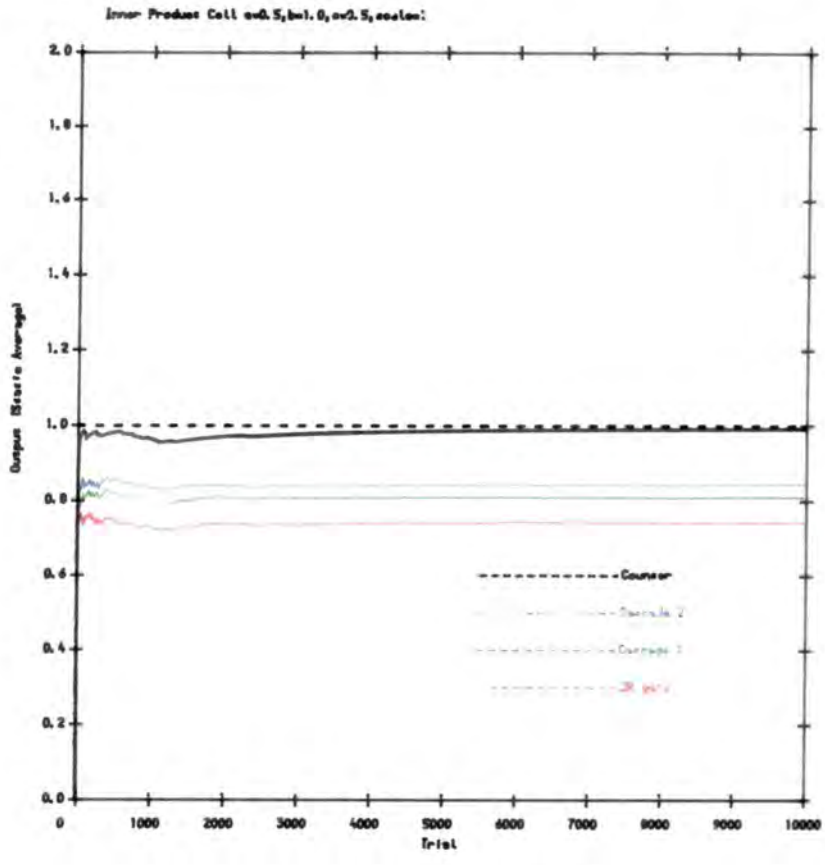


Figure 2.18a Element Output – a=0.5, b=1.0, c=0.5, scale 1

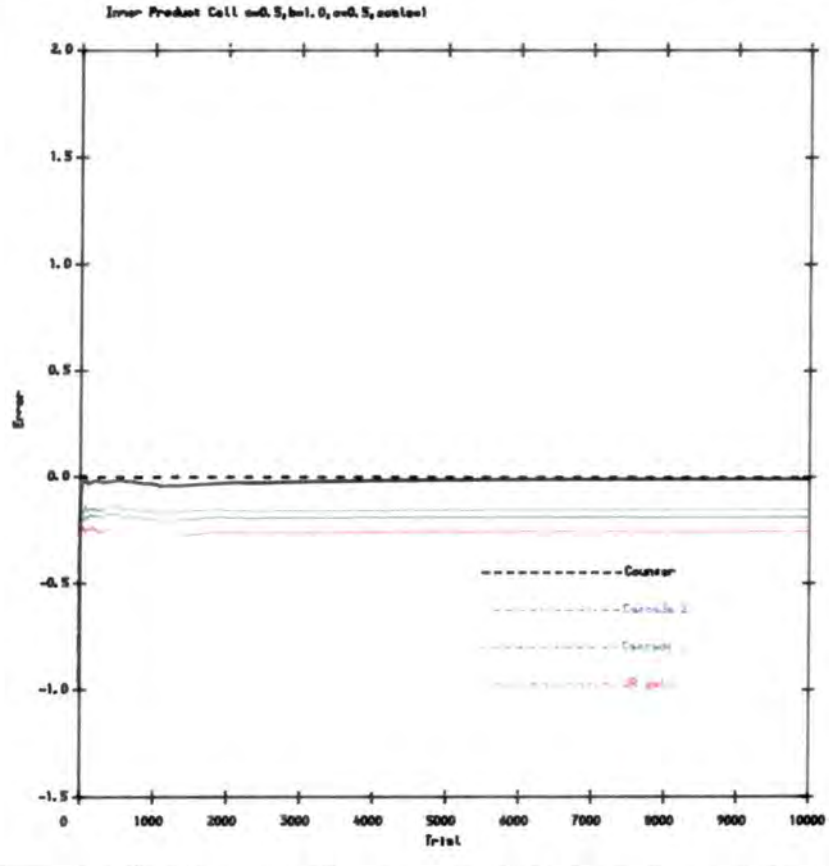


Figure 2.18b Element Error – a=0.5, b=1.0, c=0.5, scale 1

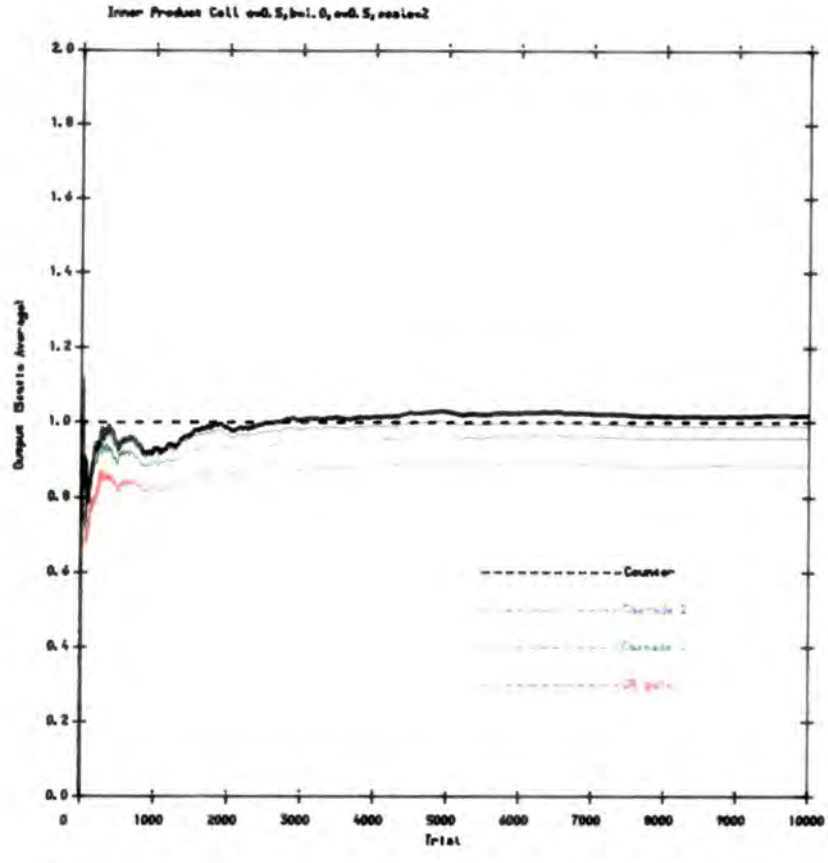


Figure 2.19a Element Output – a=0.5, b=1.0, c=0.5, scale 2

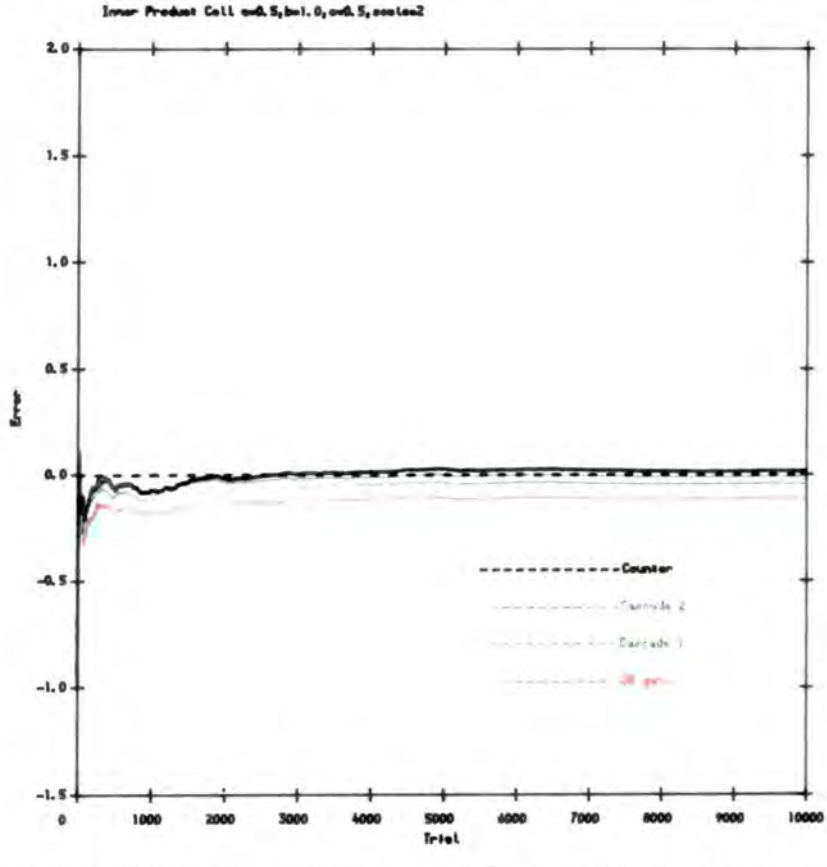


Figure 2.19b Element Error – a=0.5, b=1.0, c=0.5, scale 2

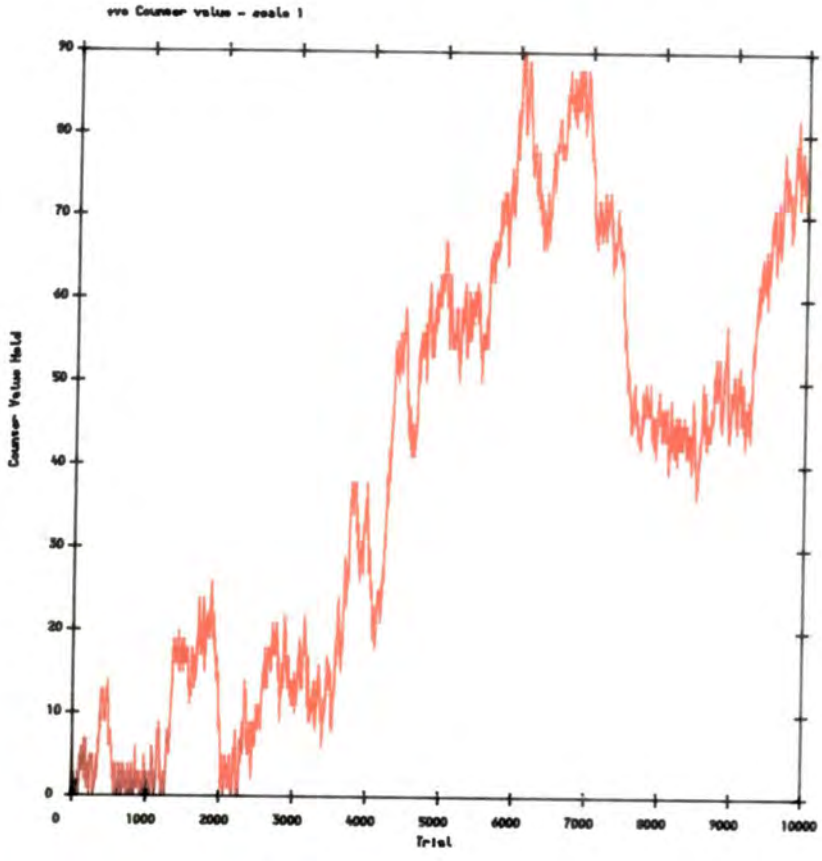


Figure 2.20a Counter State – a=0.5, b=1.0, c=0.5, scale 1

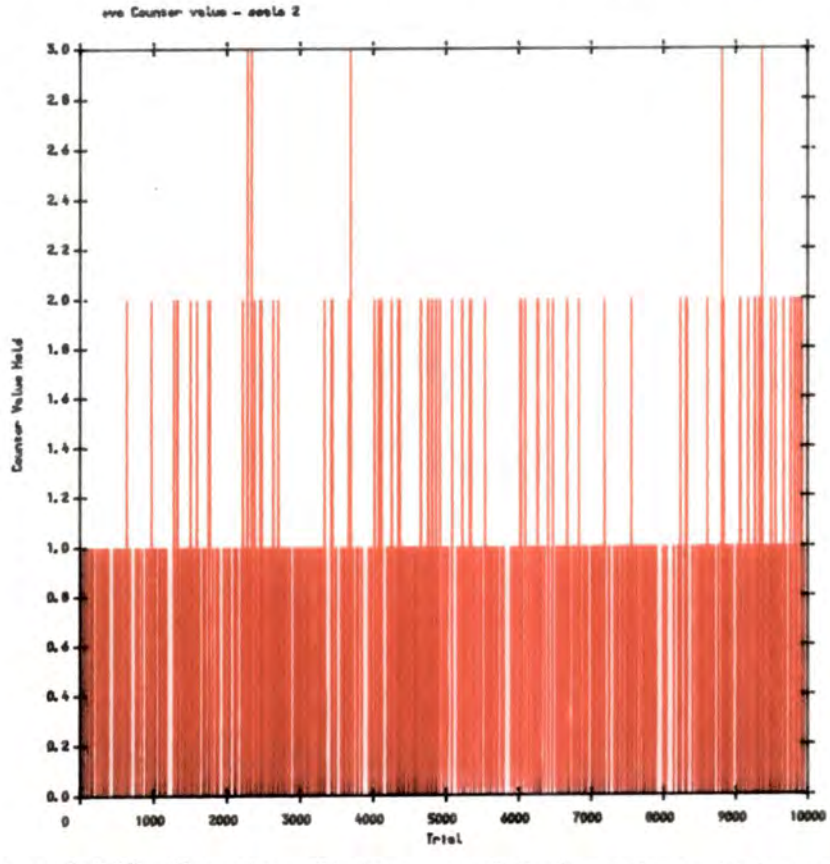


Figure 2.20b Counter State – a=0.5, b=1.0, c=0.5, scale 2

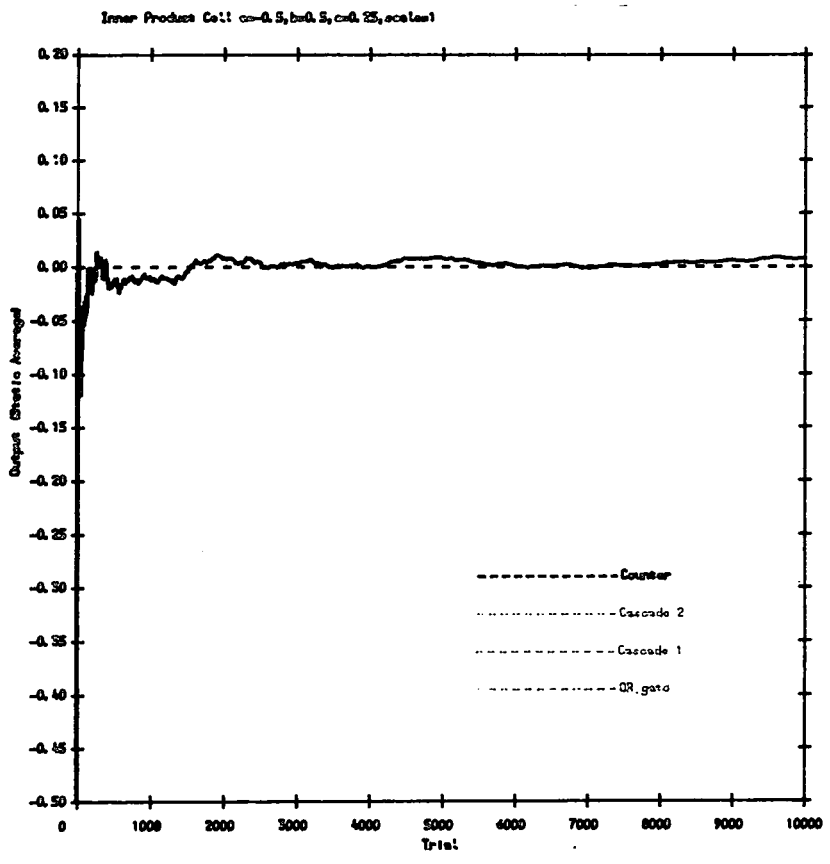


Figure 2.21a Element Output – a=–0.5, b=0.5, c=0.25, scale 1

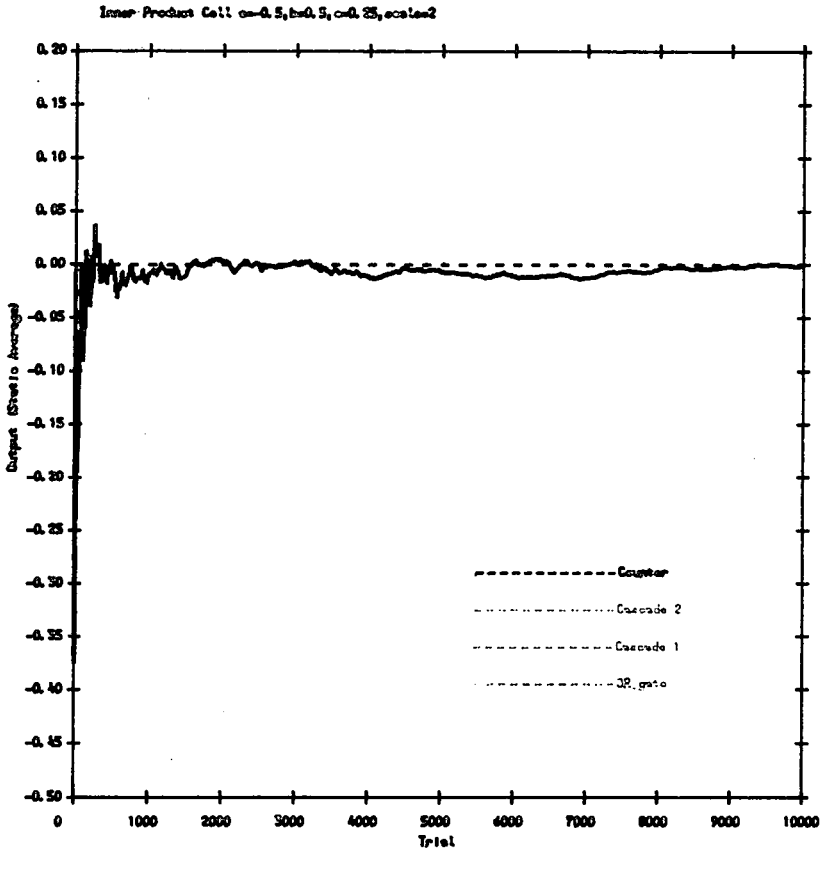


Figure 2.21b Element Output – a=–0.5, b=0.5, c=0.25, scale 2

Chapter Three

Stochastic Systolic Arrays

3.1 Introduction

This chapter examines the numerical properties of stochastic computation by extending the inner product element described in the last chapter to a novel systolic array capable of performing the Cyclic Correlation operation, [17] and thereby, for odd prime transform lengths, the Discrete Fourier Transform, [18]. The word systolic is used to describe the rhythmic heart-like pumping of data synchronously through the array, [38]. Possessing only local interconnections, the array is capable of very high speed operation, offering a fast estimation facility with accuracy dependent only on the length of the output averaging interval.

Although a digital implementation is described here, it might be possible to take advantage of the simplicity of the stochastic approach to employ some kind of fast optical logic processing, [39], [40], [41]. The speed of operation of such devices would enable fast convergence to a very accurate answer. Additionally, the stochastic correlator bit streams do not have to be precisely aligned or staggered at the inputs, as they are in deterministic processors. This relaxes timing and control requirements. Interest has been expressed in this approach for fluid flow analysis using Laser Doppler Anemometry, [12], [19] and direct input of stochastic variables such as the

outputs of photomultipliers for direct photon correlation could be possible using optical fibre connections.

3.2 Cyclic Correlation

Many applications in digital signal processing require the convolution of two sequences and in particular, the calculation of the associated correlation function. Most 'fast' algorithms, ie. those which may be performed by efficient algorithms with reduced multiplications etc. apply only to periodic functions and yield cyclic or circular convolutions and correlations, [17]. An example of a demanding application would be imaging radar signal processing such as Synthetic Aperture Radar, where large amounts of data must be correlated with reference sequences in order to extract the required information, [42], [43], [44].

As explained in Chapter Two, when using stochastic methods, multiplication becomes a trivial operation using AND gates, and so any reduction in the number of multiplications required is largely irrelevant. However care must be taken to prevent repeated multiplication of the same sequence as this attenuates the product pulse sequence dramatically. The Cyclic Correlation function is very useful for signal processing, [17], and can, for odd prime transform lengths, yield the Discrete Fourier Transform, [18].

Stochastic computation of the Discrete Fourier Transform is of particular interest for fast spectral analysis rather than frequency domain correlation/convolution methods. Consider convolution initially, for the one-dimensional case, the **Aperiodic Convolution** y of a sequence x of N terms

and the sequence h of L terms is defined as;

$$y_n = \sum_{k=0}^{L-1} h_k . x_{n-k} \tag{3.1}$$

where

$$n = 0, 1, ..., N + L - 2$$

and

$$x_{n-k} = 0$$

$$\text{if } n - k < 0$$

The aperiodic convolution of two sequences of lengths N and L results in a sequence of N+L–1 values. In contrast **Cyclic Convolution** convolves two length N sequences, giving a length N sequence as a result. Cyclic Convolution is defined as;

$$y_n = \sum_{k=0}^{N-1} h_k . x_{<n-k>} \tag{3.2}$$

where

$$\text{n} = 0,\ 1,....,\ \text{N–1}$$

and

$$< > = \text{reduction MOD N}$$

The correlation equation is the same except for <n+k> instead of <n-k>.

The remainder of this work considers the correlation function. To calculate the aperiodic correlation of two sequences of lengths L and N using a cyclic correlation algorithm requires an N+L–1 cyclic correlation. The two

input sequences are extended to length N+L-1 by appending N-1 zeroes to the h sequence and L-1 zeroes to the x sequence. Most deterministic methods take advantage of the fast algorithms available for computing the cyclic correlation function by splitting up a long correlation into a series of cyclic correlations using the standard Overlap-Add or Overlap-Save methods.

3.3 Discrete Fourier Transform

The Discrete Fourier Transform is defined by the expression;

$$X_k = \sum_{n=0}^{N-1} x_n . W_N^{nk} \tag{3.3}$$

where

$$k = 0,\ 1,\ldots,\ N-1$$

and

$$W_N = e^{-j\frac{2\pi}{N}},\ j = \sqrt{-1}$$

When N is an odd prime N_p, then the DFT may be reordered into a correlation using a technique due to Rader, [18], which operates as follows.

For k=0, X_k is simply the summation

$$X_0 = \sum_{n=0}^{N_p-1} x_n \tag{3.4}$$

For $k \neq 0$ we obtain a new sequence of length $N_p - 1$

$$X'_k = \sum_{n=1}^{N_p-1} x_n . W_N^{nk} \tag{3.5}$$

where

$$X_k = X_0 + X'_k$$

and

$$k = 1, 2, \ldots, N_p-1$$

The indices n and k are defined modulo N_p. As N_p is prime, there is a number not necessarily unique, such that there is a one-to-one mapping of the integers i=1, 2,....., N_p-1 to the integers j=1, 2,....N_p-1 given by

$$j = g^i \bmod N_p \tag{3.6}$$

The integer g is known as a primitive root of N_p. A table of primitive roots may be found in reference [45]. So for $n,k \neq 0$ we can replace n and k by the transformations

$$n = g^u \bmod N_p$$

and

$$k = g^v \bmod N_p \tag{3.7}$$

where

$$u,v = 0, 1, \ldots, N_p-2$$

and so equation 3.5 becomes;

$$X'_{g^v} = \sum_{u=0}^{N_p-2} x_{g^u} . W^{g^{u+v}} \tag{3.8}$$

where

$$v = 0, 1, \ldots, N_p-2$$

In equation 3.8 the **exponents** of g must be taken modulo N_p-1 (as opposed to the evaluation of g itself). This expression may then be rewritten as

$$\bar{X'}_v = \sum_{u=0}^{N_p-2} \bar{x_u} . h_{u+v} \tag{3.9}$$

where

$$v = 0, 1, \ldots, N_p-2$$

and

$$\{\bar{x}_u = x_{g^u}\},\ \{\bar{X}'_v = X'_{g^v}\} \text{ and } \{h_{u+v} = W^{g^{u+v}}\} \tag{3.10}$$

Equation 3.9 represents an $N_p - 1$ point circular correlation. To obtain a circular convolution would require the sign of u to be changed. This corresponds to fixing x_{g^0} and reversing the remainder of the permuted input sequence.

As an example, consider the calculation of a seven point DFT. This may be written in matrix vector notation as;

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & W^1 & W^2 & W^3 & W^4 & W^5 & W^6 \\ 1 & W^2 & W^4 & W^6 & W^1 & W^3 & W^5 \\ 1 & W^3 & W^6 & W^2 & W^5 & W^1 & W^4 \\ 1 & W^4 & W^1 & W^5 & W^2 & W^6 & W^3 \\ 1 & W^5 & W^3 & W^1 & W^6 & W^4 & W^2 \\ 1 & W^6 & W^5 & W^4 & W^3 & W^2 & W^1 \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} \tag{3.11}$$

and $W^7 = 1$.

For N=7, a suitable primitive root is 3. Thus, excluding x_0 and by using the mapping given in equation 3.6, with g=3, the input and output vectors are arranged in the 6 point sequence 1, 3, 2, 6, 4, 5 which gives;

$$\begin{bmatrix} X'_1 \\ X'_3 \\ X'_2 \\ X'_6 \\ X'_4 \\ X'_5 \end{bmatrix} = \begin{bmatrix} W^1 & W^3 & W^2 & W^6 & W^4 & W^5 \\ W^3 & W^2 & W^6 & W^4 & W^5 & W^1 \\ W^2 & W^6 & W^4 & W^5 & W^1 & W^3 \\ W^6 & W^4 & W^5 & W^1 & W^3 & W^2 \\ W^4 & W^5 & W^1 & W^3 & W^2 & W^6 \\ W^5 & W^1 & W^3 & W^2 & W^6 & W^4 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_3 \\ x_2 \\ x_6 \\ x_4 \\ x_5 \end{bmatrix} \tag{3.12}$$

and $W^7 = 1$. Equation 3.12 has the form of a Cyclic Correlation.

It is important to note by comparing equation 3.12 with equation 3.2 (and 3.13 in the next section), that the 'twiddle' complex coefficients form

the x input data sequence in the correlation and that the sequence to be transformed forms the reference sequence in the correlation.

3.4 Systolic Arrays

Systolic arrays enable very efficient parallel computation of many common processing problems, [38], [46], [47], [48]. The systolic array consists of a series of identical cells, each performing the same operation, communicating only with nearest neighbours, on each clock cycle. Since the array is regular in design with identical elements, the design and test process is simplified. Also because of the local, synchronous connections, the array may be operated at high speeds.

3.4.1 Systolic arrays for correlation

Previous work by Kung et al., [49] in the design and use of deterministic systolic arrays has resulted in a general purpose 32-bit systolic array called 'Warp', operating in conjunction with a host computer. The name refers both to its high speed operation and image processing facilities. Other work has examined the use of restricted ring or residue arithmetic, [50], [51]. By choosing special mutually prime moduli, which are powers of two, such as Mersenne and Fermat numbers, Number Theoretic transforms may be constructed which possess the Cyclic Convolution Property, [17], [50], [51], [52], allowing the use of 'fast' algorithms for correlation. It has been described how the Discrete Fourier Transform (DFT) may itself be expressed using special mappings, [18], such that it may be computed using a Cyclic

Convolution. Note also that the DFT has the Cyclic Convolution property, such that multiplication of two transforms in the Frequency Domain results in Convolution in the Time Domain. Examination of this nesting effect leads to the derivation of novel methods of performing convolution using 1–bit systolic arrays, [53], [54], [55], [56].

3.4.2 Stochastic systolic array

Equation 3.2 may be expanded for an N= 3 point correlation as;

$$\begin{aligned} y_0 &= h_0.x_0 + h_1.x_1 + h_2.x_2 \\ y_1 &= h_0.x_1 + h_1.x_2 + h_2.x_0 \\ y_2 &= h_0.x_2 + h_1.x_0 + h_2.x_1 \end{aligned} \tag{3.13}$$

It may be seen from equation 3.13 that repeated use of the same h coefficients occurs during the series summations. A suitable systolic array for correlation is depicted in Figure 3.1. The depiction of a simple rectangular cell array was chosen for computational convenience and other cell geometries, e.g. hexagonal could be more compact, [38]. The nodes have all been numbered for ease of reference. Consider applying the values h_0, h_1 and h_2 to the horizontal rows commencing at nodes 9, 5 and 1 respectively and the values x_0, x_1, x_2, x_0, x_1 to the vertical column inputs commencing at nodes 1, 3, 6, 10 and 13 respectively, (where x_0 and x_1 are repeated to generate the correct expression along the diagonals). Then it may be seen that if the action of each cell is that of the Inner Product operation as described in the last chapter, and with zero input to the diagonal connections commencing at

nodes 10, 11 and 12, the product/accumulate operation along the diagonals will produce the values y_0, y_1, y_2 at the output diagonal nodes 1,2 and 3 respectively corresponding to equation 3.13.

3.4.3 Systolic Array Simulation Results

The operation of the array has been simulated for both Complex Correlation and the Discrete Fourier Transform using the programs contained and documented in Appendix Two. Numerical Algorithm Group (NAG) library programs together with original programs were used to generate deterministic DFT and correlation results for checking purposes. Complex arithmetic using stochastically encoded variables was performed by separating input data into the four real and imaginary combinations and then using four identical stochastic arrays in parallel. The results were recombined after output averaging.

Computational overheads restricted the simulation to values of N less than 50 points. This also precluded study of multidimensional mappings of small N stochastic blocks to perform large N transforms. The output interface was the Moving Average method utilising a simulated shift register. The provision for the introduction of x dB of white or Gaussian noise with zero mean was made. The dB ratio was defined as the ratio of the standard deviation (σ) of noise added to the input values (before stochastic encoding) to the maximum input dynamic range of the array (equivalent to the square

root of the output dynamic range) ie;

$$\sigma(dB) = 20\log_{10}\{\frac{\sigma}{\sqrt{\text{OutputDynamicRange}}}\} \tag{3.14}$$

The simulation used a NAG routine to add noise to the input signal with the standard Normal (Gaussian) probability distribution given by;

$$P(X) = \frac{1}{\sigma\sqrt{2\pi}}\exp[\frac{-(x-a)^2}{2\sigma^2}] \tag{3.15}$$

In this expression, a is the mean, and corresponds to the input value with zero noise, and σ is the standard deviation. An input value of -10000dB therefore corresponds to zero noise, whilst -3dB corresponds to addition of noise with a distribution whose variance (σ^2) is half the output dynamic range.

3.4.4 6 point Complex Correlation

Figures 3.2(a,b) show the real and imaginary outputs from the array for a 6 point Complex Cyclic Correlation with zero noise. The values were chosen so that the array computed the first stage in a 7 point Discrete Fourier Transform as described previously.

The input data sequence consisted of reordered sine and cosine ('twiddle') terms as shown in Table 3.1. The reference sequence is shown in Table 3.2 and the resulting correlation as computed by a deterministic check program in Table 3.3. The system scale was chosen to be 5.0 in order to encompass the maximum output value of the correlation and interim noise.

Figures 3.2(a,b) show the value of each output point in the correlation against the number of clock cycles with -10000 dB (ie. zero) noise added

to the input. The averaging process was performed over a simulated shift register of capacity 2048 and so the averaging time interval was 2048 slots. The dotted lines indicate the theoretical answers (Table 3.3) to which the array outputs should asymptote. Note that the convergence does not improve over the run since the output averaging interval remains constant.

Figures 3.3(a,b) show an identical simulation run (using the same random number sequence) but this time with -3dB noise added to the inputs. It may be concluded from these results that the array outputs successfully converge about the expected values and that addition of Gaussian noise with zero mean to the inputs has little effect on the output. This is to be expected from the stochastic encoding and output conversion averaging processes.

3.4.5 47 point Discrete Fourier Transform

The final example presents the Discrete Fourier Transform of a 47 point sequence representing a pulse of amplitude '1' and width 4 points. The remaining points are set to '0'. The deterministically computed Discrete Fourier Transform of this sequence is shown in Figures 3.4(a,b). Figure 3.5 illustrates the comparative result of performing a standard **deterministic** Discrete Fourier Transform on the input data sequence, with -3dB noise added. In a practical application with this much noise, the inputs would need to be averaged (thus removing Gaussian noise with zero mean) before performing the transform and so the figure shows the results of averaging the inputs over 1, 10, 100, 1000 and 10000 snapshots before performing the transform. Figure 3.6 shows the Real and Imaginary output of the stochastic

systolic array for this input sequence with -10000 dB noise. The input range is scaled to 4.0 and snapshots taken when N, the number of trials is N=1, 10, 100, 1000 and 10000 trials. The static averaging method was used, ie. not using a shift register and the X_0 and the x_0 terms were introduced deterministically as appropriate. This can be seen clearly on the N=1 plot. The expected transform result as computed deterministically is overlaid using the dotted line.

These runs were then repeated for -3dB noise and the results plotted in Figure 3.7. It may be seen from these results that the array is relatively insensitive to noise and that a good estimate of the transform takes shape very quickly.

3.5 Dynamic Range and the Autocorrelation Function

It has been described how a Cyclic Correlation may be performed by a systolic array. By rearranging the input and reference data as explained, then the Discrete Fourier Transform may be performed. It is useful to be able to predict the maximum value for a correlation, in order to be able to scale the inputs correctly to avoid overflow. The maximum value will never exceed that given by the autocorrelation function of the reference, ie. the correlation of the reference against itself.

Since the reference is often packed with zeroes to avoid periodic repeating and overlapping of the correlation function, [17], the autocorrelation is relatively small compared to the potential worst case of a reference function such as a dc level with maximum value at all points.

3.6 Discussion - Chapter Three

The results show that the Inner Product cell described previously may be built up into a novel array to compute a useful signal processing function. Comparisons with deterministic methods can be misleading due to the fundamental differences between conventional digital computation and stochastic (or digital-analogue) computation. Bearing this in mind, it is still possible to draw some conclusions.

3.6.1 Array accuracy

The array is inherently noise insensitive but has relatively low accuracy compared with standard deterministic methods. As long as the output sequences remain close to Bernoulli, then the distribution of estimates about the mean is as described by equation 2.2 in Chapter Two. Note however that the effective accuracy of the system when used to perform the DFT might be partially improved, depending on the data values due to the mixed deterministic/stochastic nature of the computation. The system could provide fast Spectral Analysis in noise where accuracy would be inversely proportional to bandwidth.

3.6.2 Array size

If a large array were constructed, to perform a 101-point Discrete Fourier Transform, it would consist of 10,000 cells by four arrays, ie. 40,000 cells (for the 100-point Cyclic Correlation required). If one realistically constructs 1000 such cells on a chip (actually 31 x 31), then 40 chips would

be required which could be mounted on one small board. There would therefore be about 100 boards for the 1031-point DFT and at this point, Wafer Scale Implementation starts to look attractive. The array computes a length N transform using an array comprising N^2 cells. Whilst the cell design is reasonably simple and compact as described in Appendix 3, this still represents a considerable chip area and pin count limitation. Consider Figure 3.1; each cell is performing the Inner Product operation and the total sum is formed at the conclusion of each diagonal. It may be seen that an alternative scheme would be to merge the operations of cells 1, 4, and 7 together into one new cell, similarly for cells 2, 5, and 8 and 3, 6, and 9.

Consider a deterministic scheme in which the three cells are in a line. Processing takes place by passing the values along the line in a sequential or pipelined fashion, storing any necessary values along the way. In this way, it would be possible to construct pipelined bit-serial and bit-parallel deterministic alternatives to the array which have the same accuracy and yet operate in much reduced chip areas and operation cycles, with much higher throughput. Reference [57] gives a good description of such arrangements. A stochastic implementation using N cells would require that each cell has 2N inputs and a single output for an N length transform where the inputs are effectively shared between cells. In this case, the cell could be considered to be forming a weighted summation of N input stochastic sequences where the weights are formed by the reference sequences. This concept is considered later in the thesis in connection with neural networks.

3.6.3 Array speed

The array throughput (ie. time between successive output vectors) is low due to the need to take a time average, also no pipelining is possible. This might not necessarily be a problem when processing data in a noisy environment where a deterministic system would need to average over many inputs before performing the required computation. Furthermore, only comparators are necessary at the inputs rather than analogue to digital converters. This could increase speed and decrease the hardware cost. No cell is idle and the local synchronous single bit connections allow the array to operate at high speeds whilst allowing for fault tolerance to be built in. If the array operated at a modest 50MHz, then the mid-range accuracy would be 10% in 2μsecs, 1% in 0.2msec and 0.1% in 0.02sec. This should be independent of the size of transform and is true as long as the sequences remain close to Bernoulli. It is difficult to give 'best' figures for deterministic processors, application specific designs are always likely to be faster than general purpose machines and input/output considerations may be necessary. However, as a general guide, recent figures describe 16 bit 1024 point DFTs in both 0.5ms [58] and 0.021ms [59].

3.7 Conclusions and Summary – Chapter Three

The main conclusion to be drawn from this work is to show that the stochastic array, consisting of simple, compact stochastic elements may operate on single bit input probability streams to compute a useful mathematical function. The Inner Product cells described effectively perform weighted

summation using stochastic encoding and this is an important result which is explored in later chapters concerning neural networks.

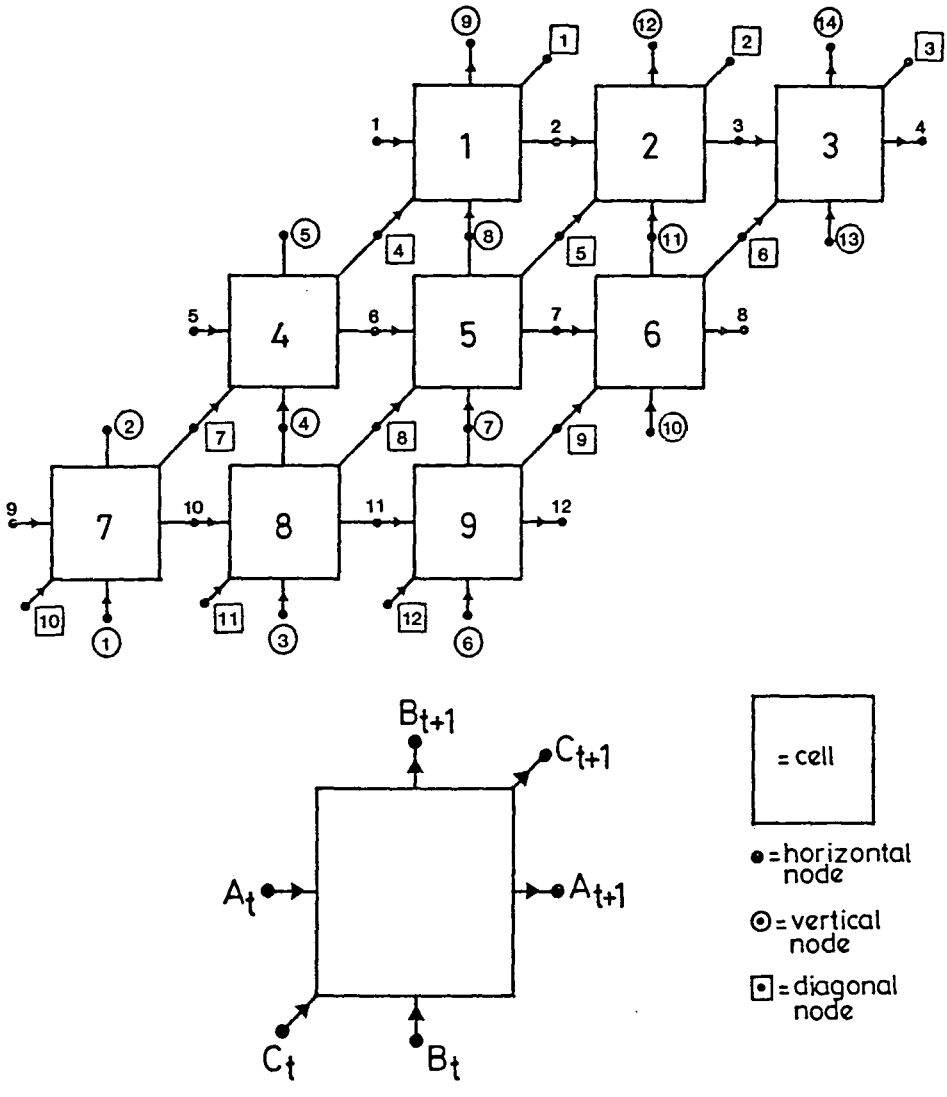


Figure 3.1 Systolic Array for Cyclic Correlation

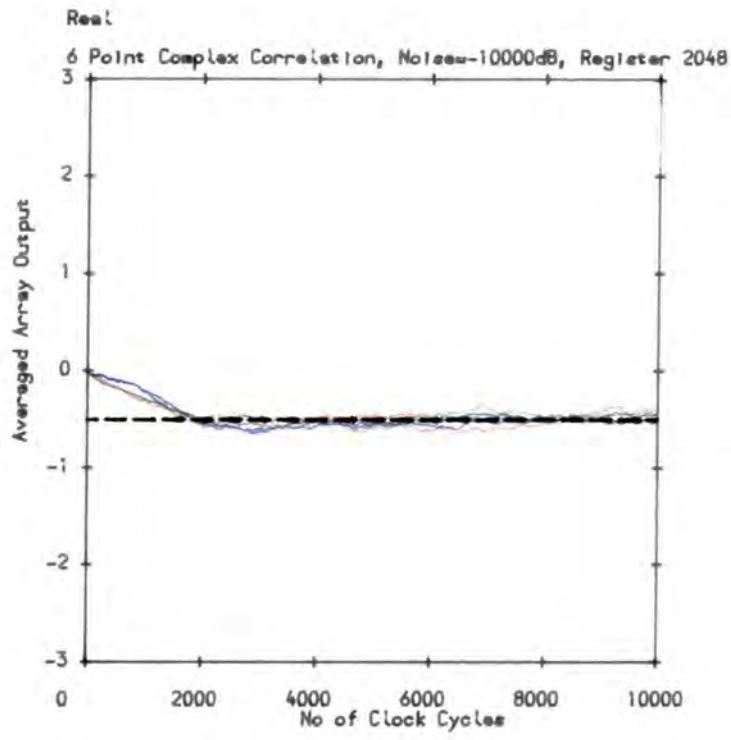


Figure 3.2a Array output (Real) - Noise -10000 dB

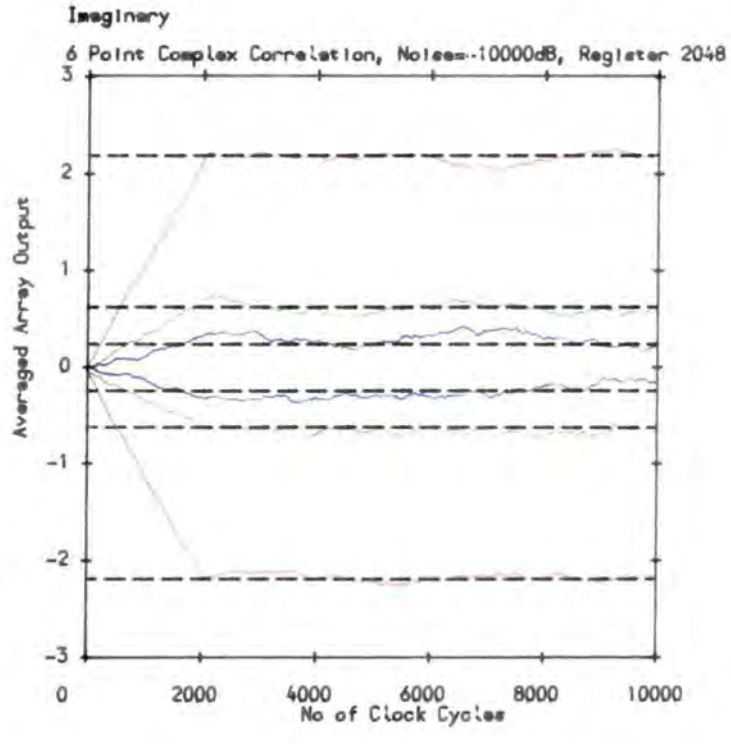


Figure 3.2b Array output (Imag.)- Noise -10000 dB

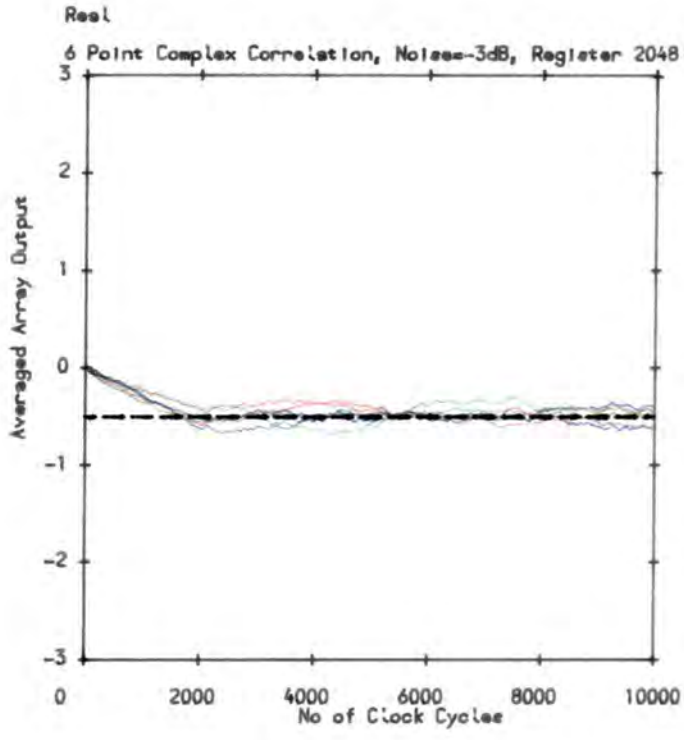


Figure 3.3a Array output (Real) – Noise –3 dB

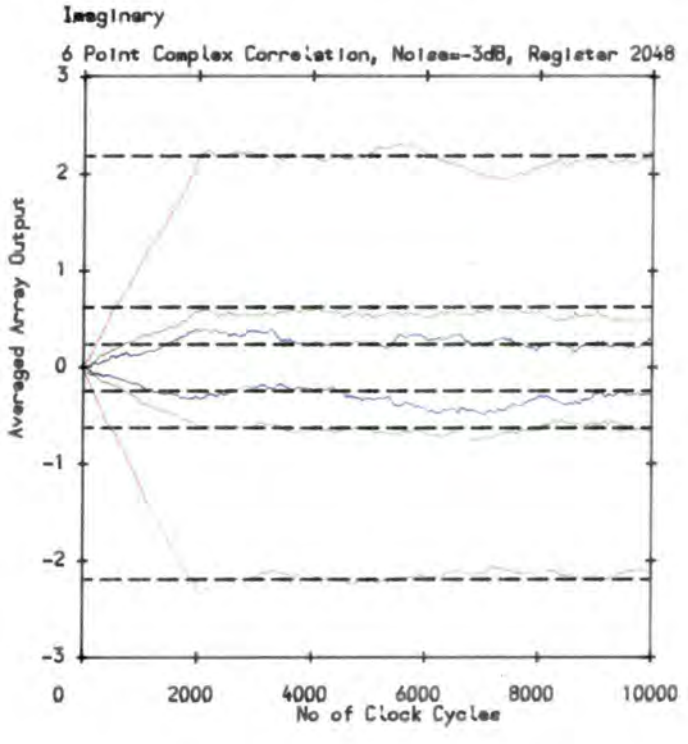


Figure 3.3b Array output (Imag.)– Noise –3 dB

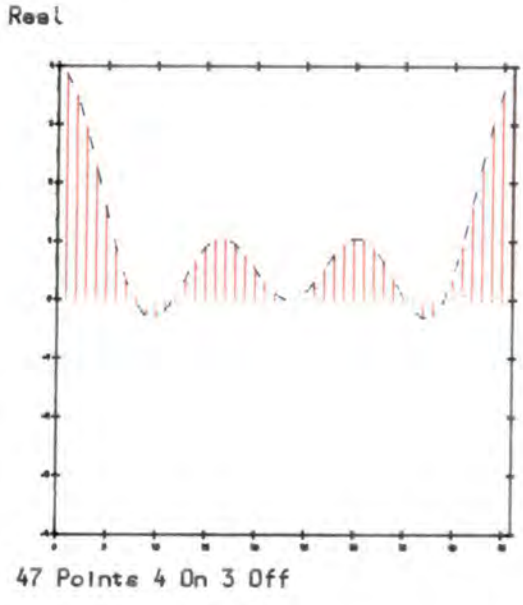


Figure 3.4a 47 point DFT result Real output

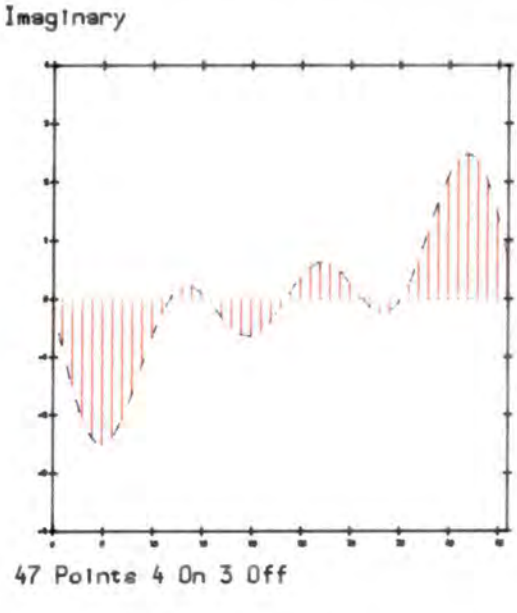


Figure 3.4b 47 point DFT result Imag. output

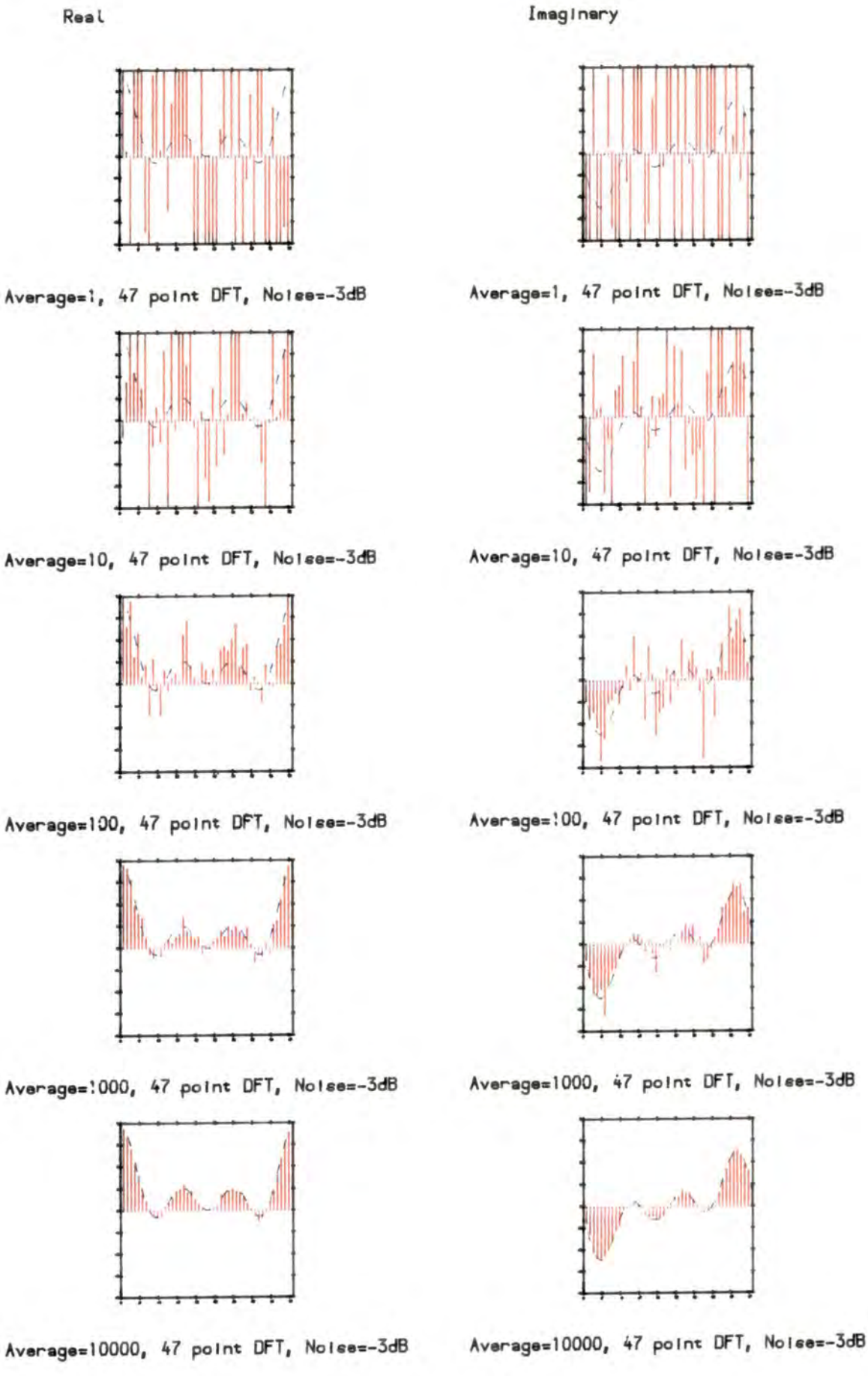


Figure 3.5 47 point DFT results Real & Imaginary output – Noise –3dB

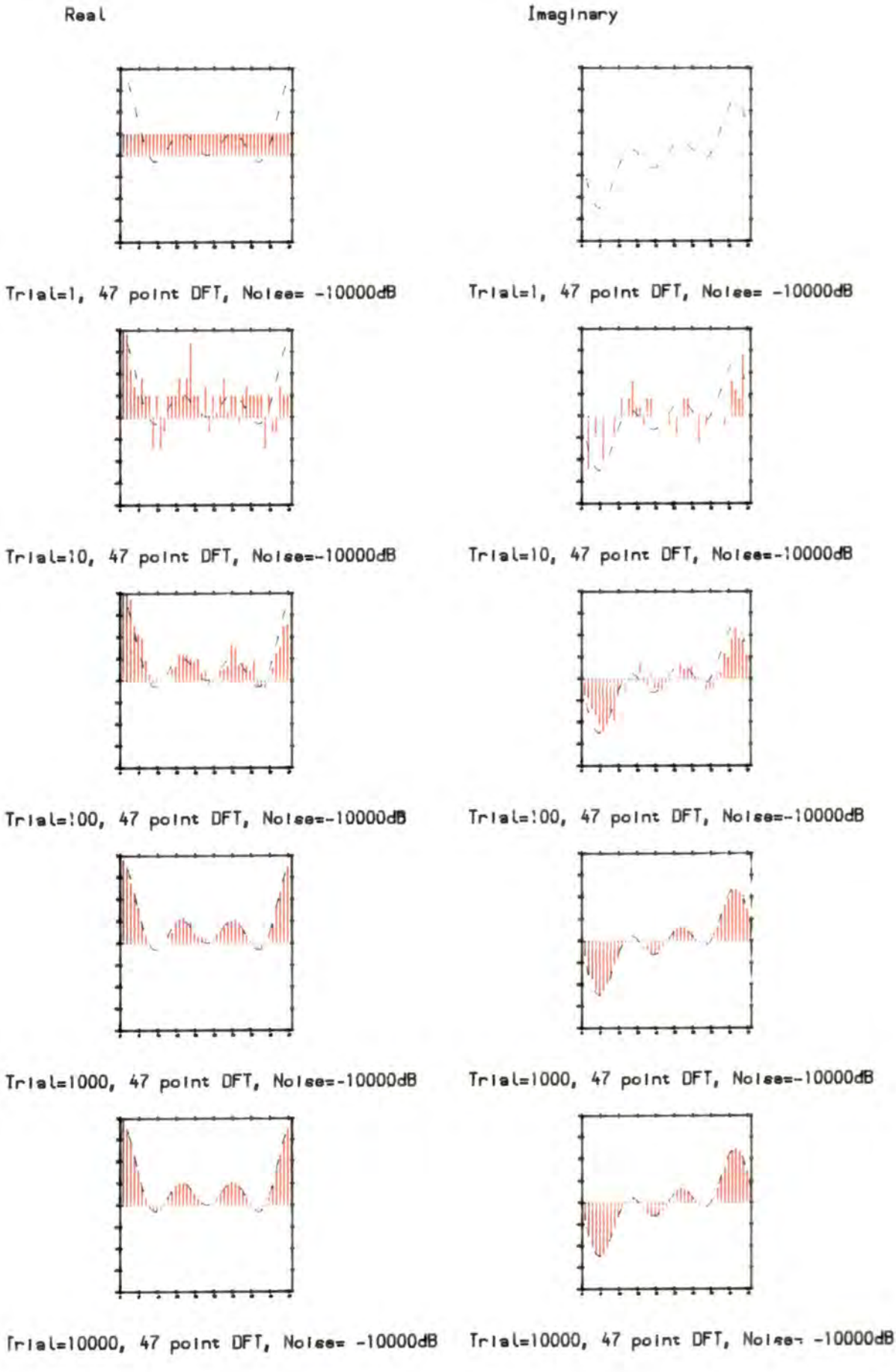


Figure 3.6 47 point DFT Real & Imaginary array output – Noise –10000 dB

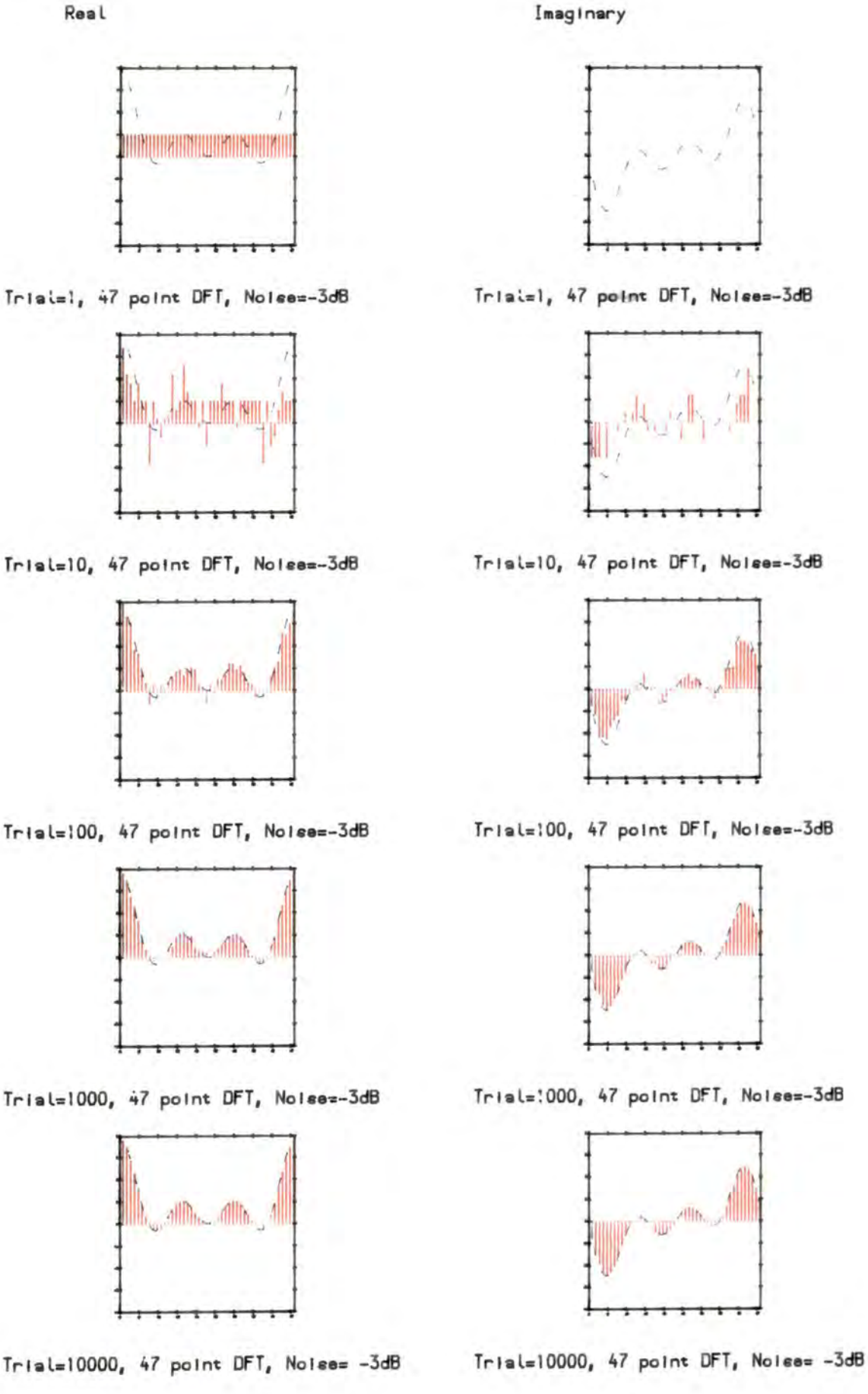


Figure 3.7 47 point DFT Real & Imaginary array output – Noise –3 dB

Table 3.1: Reordered Sine and Cosine Terms (x sequence)

Real	Imaginary
0.6234904	−0.7818310
-0.9009672	−0.4338870
-0.2225188	−0.9749284
0.6234863	0.7818343
-0.9009703	0.4338807
-0.2225256	0.9749269

Table 3.2: Reordered input data sequence (h sequence)

Real	Imaginary
1.0	0.0
1.0	0.0
1.0	0.0
0.0	0.0
0.0	0.0
0.0	0.0

Table 3.3: Resultant Deterministic Correlation

Real	Imaginary
-0.4999956	−2.190645
-0.4999993	−0.6269811
-0.5000028	0.2407866
-0.5000096	2.19064
-0.5000055	0.6269765
-0.5000024	−0.2407904

Chapter Four

Stochastic Learning Automata

4.1 Introduction

The previous two chapters described how simple stochastic cells, operating locally, could interact with single bit probabilistic inputs to compute a useful result in the field of digital signal processing (DSP) using a stochastic network. The use of such cells is not restricted however to DSP. Similar elements have been developed for successful use in optimisation and control problems where there is little or no prior knowledge of the process required. These stochastic elements can interact with an unknown environment and adapt their 'performance' according to some arbitrary criterion to a better than random response. For this reason they are termed stochastic learning automata, [20], [60]. These automata effectively use past experience and interaction with a random environment to optimise their response to external factors. Some work has been prompted from the study of biological systems, [61].

This chapter considers stochastic automata both singly and as part of larger learning arrays. This theme of learning arrays will be combined with that of stochastic networks in the later chapters on neural networks. The concept of self-organising machines which learn by simple feedback with an outside agency or environment is attractive since it removes the necessity for

explicit instruction. As mentioned, a characteristic of the learning automaton is the optimisation of its response in some fashion to an external environment. By suitable choice of the environment, the automata may find the global optimum of an expression not necessarily known in advance.

There are two main methods for performing global optimisation, [20], [62]. The first involves the reduction of the problem to the determination of an optimal set of parameters and then performing some kind of gradient following technique to find this optimum. The second method is by reducing the problem to the choice of the optimal action out of a set of allowable actions and this latter method is used by learning automata.

4.2 Interaction Learning

The basic structure of the learning automaton approach consists of a stochastic automaton–environment interaction as shown in Figure 4.1. The stochastic automaton performs an action which is input to the environment. The environment then outputs a response which is in turn input to the automaton in a feedback arrangement. This response indicates whether the automaton action was correct, note though that the environment response itself may be incorrect, although this is required to occur with a lesser probability than the correct response. The goal of the automaton is to perform the action which corresponds to the maximum or minimum of some performance criteria. Initially, the probability of selecting any of the available actions is equal. The automaton learning process involves updating the probability of selecting such an action in order to select the optimum action and there are

a number of algorithms possible.

The one major application where automata have been found to have near optimal performance is in the area of Communication Networks, [13], [14], [63]. This involves a stochastic automaton at each node of a network which chooses paths to route calls as its actions as shown in Figure 4.2. If a call is successful then the automata which took part in the routing are in some way credited by their environment, enabling them to update their action probabilities in a way which should increase their chances of selecting such a move again. If the call is blocked, then similarly their learning rule will seek to reduce the probability of selecting that path again. This is of particular interest since there is minimisation of a global performance index, namely the Blocking Probability (Circuit–Switched Networks) and Average Packet Delay (Packet Networks) but from purely local updates and decentralised control. Secondly, if the network is damaged or spare capacity becomes available, there is a rapid response from the elements to adapt to this and move to a new optimal routing configuration.

Stochastic learning automata are often implemented in software, but by their nature are also cheap to construct since by using stochastic encoding techniques, they may be designed using simple hardware such as AND and OR gates and counters, [64], [65], [66], [67], [68], [69].

4.3 Stochastic Automaton

Figure 4.3 shows a stochastic automaton. It may be considered as a sextuple, $\{X, \phi, \alpha, p, A, G\}$, [20]. X is the input set consisting of allowable

inputs to the automaton as its elements and is typically the output from the environment.

There are three possible types of environment. The first is known as P–type; for an automaton operating with a P model environment, the set has two input elements, $x_1 = 0$ and $x_2 = 1$. In other words, a binary input $X = \{0, 1\}$. The second model is known as the Q–type; in this case, the elements are discrete and chosen between the values 0 and 1. Therefore $X = \{x_1,\ x_2,\ x_3, \ldots\ x_k\}$ where $x_i \in [0, 1]$. The third model is known as the S–type; this allows the elements to have a continuous rather than discrete range of values between the limits of 0 and 1.

Consider the P–type model, which is the main model adopted for the work described in this thesis. The input connection to the automaton from the environment is termed the reward/punish line. The usual convention is that the presence of a 1 implies a punish or penalty signal and the absence of a 1 is taken to be a reward or non–penalty signal although this is arbitrary and other similar mappings are often used e.g. [1, –1] or [1, 0] rather than [0, 1].

The set of internal states of the automaton is represented by $\phi = \{\phi_1,\ \phi_2,\ \ldots, \phi_s\}$ and the output or action set is $\alpha = \{\alpha_1,\ \alpha_2,\ \ldots, \alpha_r\}$ with $r \leq s$, p is the state probability vector governing the choice of the state at each stage, $p(n) = (p_1, p_2,\ \ldots, p_n)$. A is the automaton's algorithm, also known as an updating or reinforcement scheme. This describes its operation generating p(n+1) from p(n). G is is the output function of the automaton which relates the state of the automaton to the output. $G : \phi \rightarrow \alpha$ This

may be deterministic or stochastic.

4.3.1 Environment

The random process, system or medium in which the learning automaton operates is known as the environment. The environment as shown in Figure 4.4 is defined by the triple, $\{\alpha, C, X\}$.

The environment has inputs $\alpha(n) = \{\alpha_1, \alpha_2, ..., \alpha_r\}$ and outputs or responses belonging to the set X as discussed in the last section. The probability of emitting a particular response at time t=n depends on the input received from the automaton and is denoted by $c_i(n)$ (i = 1,...., r). The $c_i(n)$ are called the penalty probabilities and are defined as

$$c_i(n) = P(x(n) = 1|\alpha(n) = \alpha_i) \tag{4.1}$$

Therefore c_i represents the probability of a penalty being output in response to the input α_i while the probability of a reward is $1 - c_i$.

It is assumed that the c_i are unknown by the automaton initially. If they do not depend on n, the environment is said to be stationary, otherwise they are non-stationary. A switched environment is one example of a non-stationary environment, where one or more penalty probabilities change instantaneously from one value to another, [63]. The environments described so far have been autonomous, in that the penalty probabilities associated with an action were unaffected by the operation of the automaton. In many practical situations, this would not be the case and a good example would be in the routing of telephone calls in a communication network where the

automaton actively affects the environment. This type of environment is therefore described as non–autonomous.

In autonomous environments, only one single action corresponds to the minimum penalty probability (or maximum reward probability) and so the performance of the automaton can be judged from how nearly optimal the automaton is. In the non–autonomous environment, since the penalty probabilities vary as the action probabilities change, no single action probability can be described as being best, and so the task of the automaton changes from finding the best action to finding the best distribution of actions. In non–autonomous environments, the degree of optimality is therefore not an effective measure of performance and the average penalty received by the automaton is used.

4.3.2 Performance

The automaton is considered to have learned, if its performance in some way improves during its interactions with the environment. A useful measure for judging the performance of the learning automaton is the average penalty received. At a certain stage n, if the action α_i is selected with probability $p_i(n)$, the expected penalty is;

$$\begin{aligned} M(n) &= E\{x(n)|p(n)\} \\ &= \sum_{i=1}^{r} p_i(n)c_i \end{aligned} \tag{4.2}$$

If no previous knowledge is assumed, and the actions are randomly chosen with equal probability, the value of the average penalty, M_o is given by;

$$M_o = \frac{c_1 + c_2 + ... + c_r}{r} \tag{4.3}$$

If the automaton is learning, then the asymptotic average penalty is less than M_o since it is behaving better than pure chance. This behaviour is known as expediency, defined as follows;

Definition 1: A learning automaton is called *expedient* if

$$\lim_{n\to\infty} E[M(n)] < M_o \tag{4.4}$$

The expectation operator is used since the problem is in general, dealing with random variables. When the automaton is expedient, it only does better than one which chooses actions randomly. If the average penalty is minimised by the best selection of actions then the learning automaton is called optimal. From equation 4.2 it can be seen that the asymptotic minimum value of M(n) is the minimum value of the penalty probabilities, c_i when chosen with probability one.

Definition 2: A learning automaton is called *optimal* if

$$\lim_{n\to\infty} E[M(n)] = c_{min}$$

where

$$c_{min} = \min_i\{c_i\}. \tag{4.5}$$

Optimal performance is not always achievable and in such a case, one would try to achieve a good sub–optimal performance. Such a property is given by ε–optimality.

Definition 3: A learning automaton is called ε – *optimal* if

$$\lim_{n\to\infty} E[M(n)] < c_{min} + \varepsilon \tag{4.6}$$

can be obtained for any arbitrary $\varepsilon > 0$ by a suitable choice of the parameters of the reinforcement scheme. ε-optimality implies that the performance of the automaton can be made as close to the optimal as desired. This is said to be conditional if these properties hold only when the values of the penalty probabilities, c_i satisfy certain restrictions, e.g. that they should lie in certain intervals.

The performance would be even better if the decrease of $E[M(n)]$ is monotonic, ie. if the expected next value of M(n) was less than the current value as follows.

Definition 4: A learning automaton is called *absolutely expedient* if

$$\lim_{n \to \infty} E[M(n+1)|p(n)] < M(n) \tag{4.7}$$

If $M(n) \leq M_o$ initially, absolute expediency implies expediency and is therefore a stronger requirement on the learning automaton.

4.4 Reinforcement Schemes

The crucial factor which affects the performance of the learning automaton is the reinforcement scheme for updating the action probabilities. In general, a reinforcement scheme can be represented by;

$$p(n+1) = T[p(n),\ \alpha(n),\ x(n)] \tag{4.8}$$

where T is an operator; $\alpha(n)$ and x(n) represent the action of the automaton and the input to the automaton from the environment at instant n, respectively. The reinforcement schemes can be classified using the learning automaton

property, e.g. expediency or optimality, or on the basis of the nature of the functions appearing in the scheme (e.g. linear, non-linear or hybrid). If p(n+1) is a linear function of the components of p(n), the reinforcement scheme is said to be linear, otherwise it is non-linear. In some cases, p(n) is updated according to different schemes depending on the intervals in which the value of p(n) lies. In such a case the combined scheme is known as hybrid. If the learning automaton selects an action α_1 at instant n and a nonpenalty (reward) input occurs, the probability of action, $p_i(n)$ is increased and all other action probabilities are decreased. For a penalty (punish) input, $p_i(n)$ is decreased and the other action probabilities are increased. Some schemes retain the unsuccessful action probabilities at the previous values, this is known as inaction. A successful reinforcement scheme adopted for use in communications network routing is known as the Linear Reward-Penalty or L_{R-P} method.

4.4.1 Linear Reward–Penalty Reinforcement Scheme

This section considers one particular reinforcement scheme known as the Linear Reward-Penalty or L_{R-P} method, [20]. When there is a reward signal then for the successful action i,

$$\begin{aligned} P_i(n+1) &= 1 - \sum_{j \neq i} AP_j(n) \\ P_j(n+1) &= AP_j(n) \end{aligned} \tag{4.9}$$

for all j $\neq$ i

where $0.0 < A < 1.0$

When there is a punish signal then;

$$P_i(n+1) = BP_i(n)$$
$$P_j(n+1) = P_j(n) + \frac{(1-B)P_i(n)}{m-1} \qquad (4.10)$$
$$\text{for all j} \neq \text{i}$$

$$\text{where } 0.0 < B < 1.0$$

$$\text{m} = \text{total number of actions}$$

$$\text{A} = \text{Learning parameter}$$

$$\text{B} = \text{Penalty parameter}$$

Equations 4.9 and 4.10 describe how the probabilities of selecting the appropriate actions are adjusted so that if successful, they are selected with greater probability, otherwise with less. Also note how the probability measure is preserved.

By setting B = 1, the Linear Reward-Inaction or L_{R-I} method is obtained which effectively ignores penalty inputs from the environment. The L_{R-P} scheme is expedient in all stationary random environments and the L_{R-I} method is ε–optimal, [20]. It is suggested from the literature that most other schemes are inferior to the L_{R-P} and L_{R-I} methods, [20].

4.5 Interaction of Automata

This chapter has so far examined single stochastic automata interacting with random environments. The real potential of such simple, probabilistic elements becomes apparent when considering interactions between several automata in learning arrays.

Two types of interaction are of particular interest. The first is when several automata are either competing or cooperating together in an environment and this is known as a game situation. This will be shown to be important in the later chapters on neural networks using cooperating automata. The second is when the automata are operating on various levels with interaction between the different levels in a hierarchical situation. Variations of this approach will also be considered in connection with neural networks.

4.5.1 Automata Games

A game between two automata is when each automaton can affect the penalty probability received by the other automaton. Each automata selects its actions without any knowledge of the operation of the other automaton. There are two types of game; competitive games, where, for each pair of actions selected by the automata, the environment responds in a random manner, with responses to the two automata which sum to zero, as shown in Figure 4.5, and cooperative games, where the automata receive the same penalty probability and can therefore cooperate to receive the lowest average penalty (highest reward). This latter situation is also known as a team of automata as shown in Figure 4.6.

The action probabilities of the two automata are then independently updated according to suitable reinforcement schemes and the procedure is repeated. In the automata games, no prior knowledge of the game or number of players is available and the automata have to choose their strategies on-

line. Automata games are often used to compare and test differing types of automata to determine the desirable qualities for automata. The use of automata in communications routing is an example of a cooperative game of automata.

4.5.2 Multilevel Automata

The use of a single two-state automata as previously described is considerably limiting for interesting problems and a multi-action automaton would be preferred. However, the hardware necessary to implement such an automaton grows rapidly. By using a multilevel tree of small-state automata, consisting of several levels, each comprising many automata where the action of each automaton at a certain level triggers automata at the next lower level and so on as shown in Figure 4.7, this problem can be overcome.

For instance, an m-level system, based on an r-state cell results in an automaton of r^m states, [70]. Each learning cycle consists of m decisions between r states, i.e. a total of r x m decisions, rather than a single level decision between r^m states but the hardware savings are considered to exceed the serial time penalty. The main problem in such multilevel systems is the coordination of activities at different levels to ensure convergence of the entire network towards the desired behaviour. By examining the decision route taken through the tree, the appropriate reinforcement or credit assignment may be made.

4.6 Conclusions and Summary – Chapter Four

This chapter has introduced the basics of stochastic learning automata. It has been described how these simple elements may interact with a random environment to improve their responses using various learning mechanisms.

It will be shown in a later chapter, how simple two-state automata may be configured to perform pattern recognition, and how they may subsequently be incorporated into larger, cooperative learning arrays or neural networks. This will be shown to be of particular interest in large decision-making problems such as pattern classification, by using a massively interconnected, parallel, multi-level learning array.

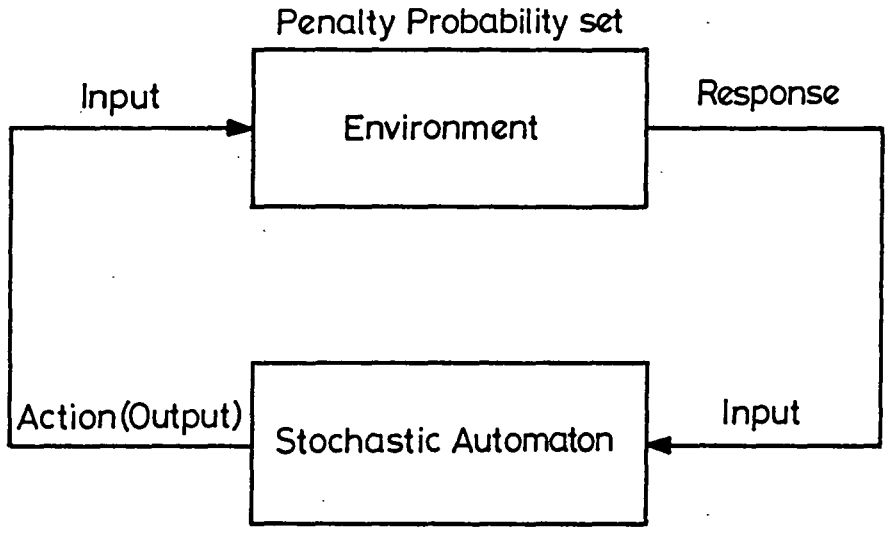


Figure 4.1 Learning Automaton

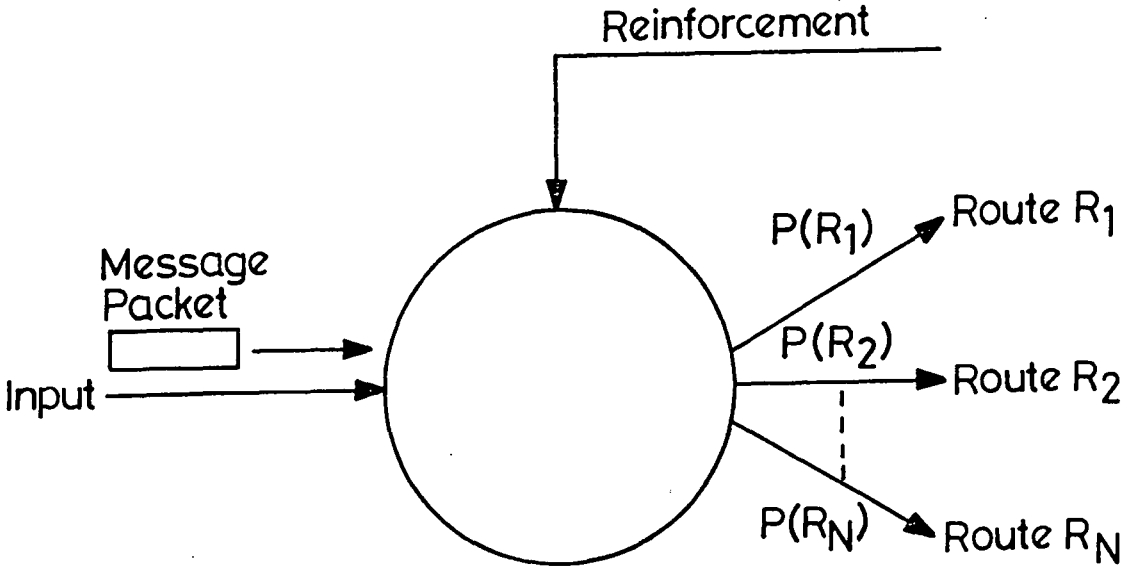


Figure 4.2 Communications Routing using Automata

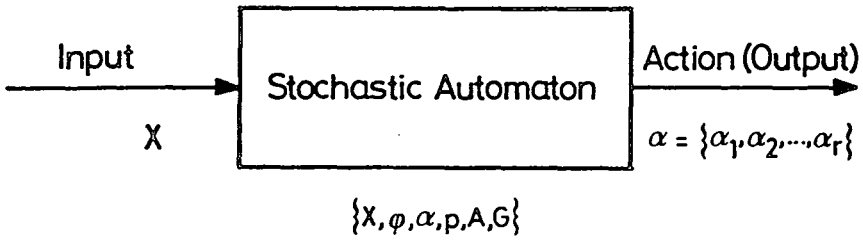


Figure 4.3 Stochastic Automaton

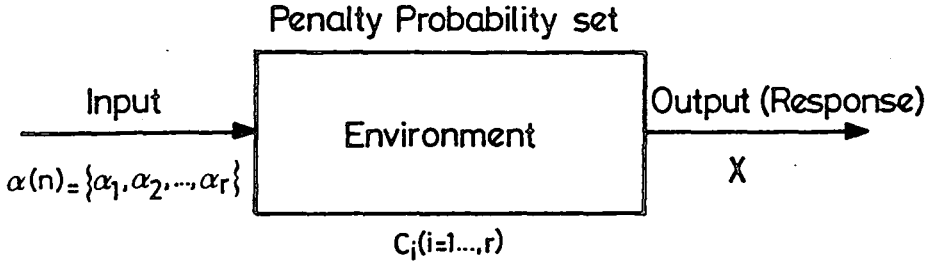


Figure 4.4 Environment

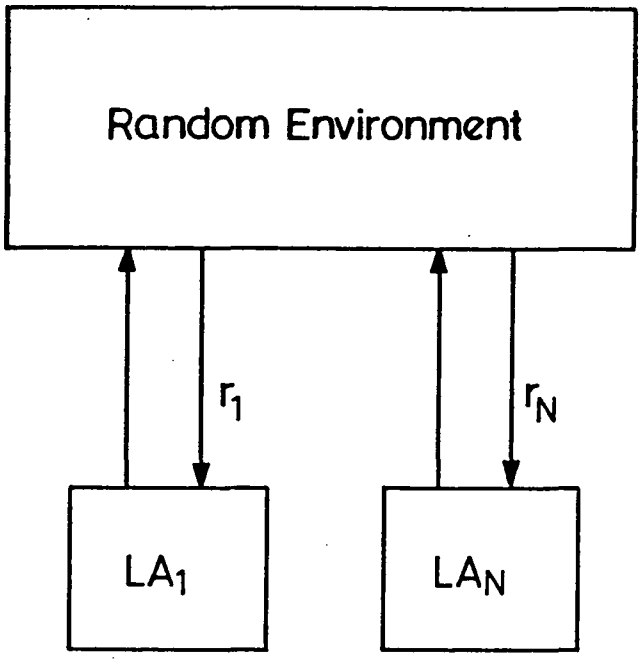


Figure 4.5 Multiple Learning Automata – Game

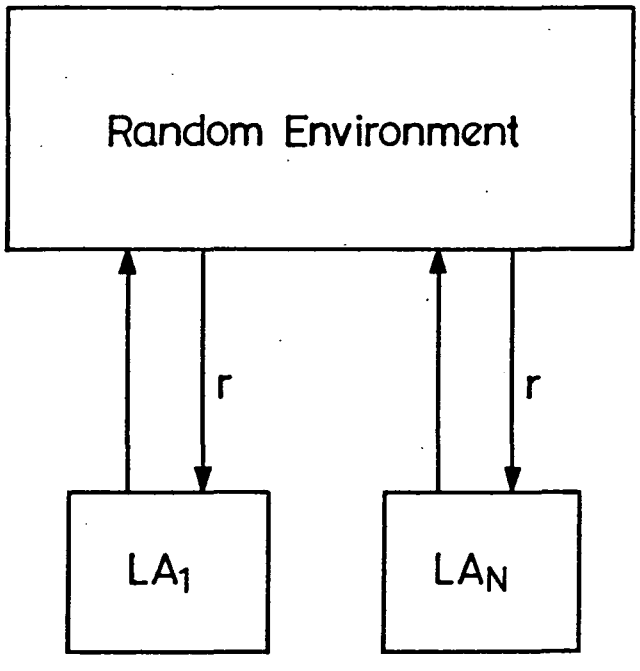


Figure 4.6 Multiple Learning Automata – Team

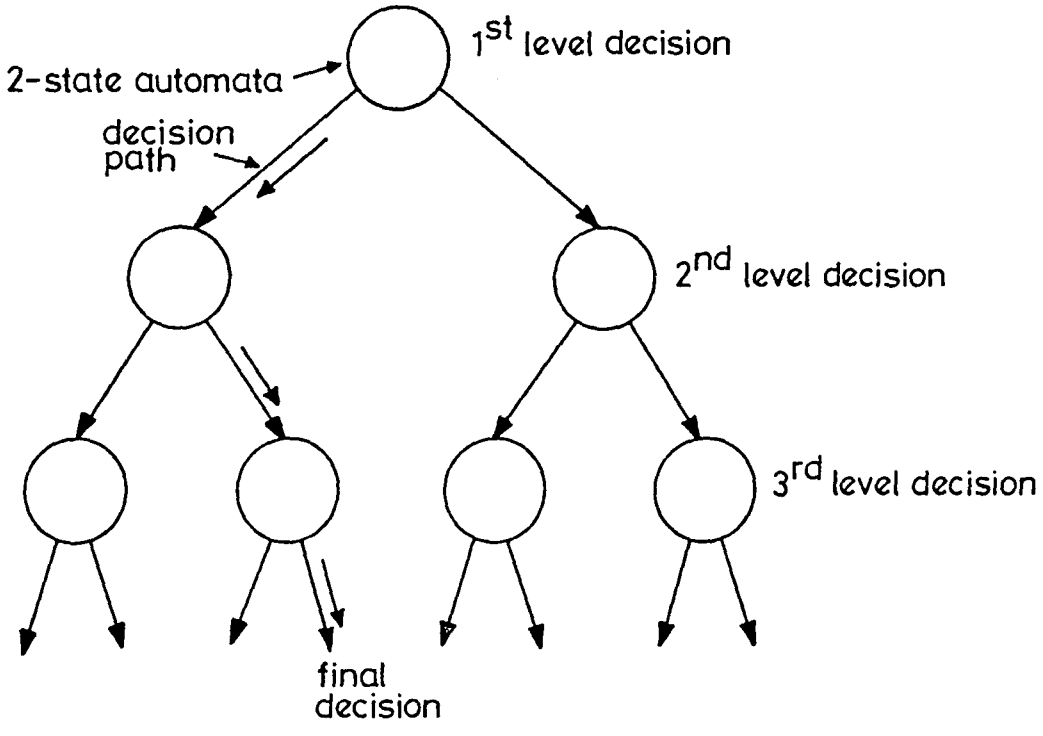


Figure 4.7 Hierarchical Learning Automata

Chapter Five

Neural Networks

5.1 Introduction

The previous chapters have described two types of stochastic element which can perform useful tasks including finding global optima, collective decision–making or operations such as transforms. The common theme throughout is one of simple probabilistic cells, noise–tolerant and yet interacting with a random environment to produce a useful result.

This final chapter of the first section of the thesis considers a simple stochastic element which has excited much activity in recent years and is modelled on the neurons of organic brains. Although neurons are thought to be similar between species, attention is particularly focussed on those found in the human brain. It will be shown that synthetic neurons modelled loosely on features extracted from their biological counterparts may be configured in learning arrays to provide such functions as pattern recognition, associative memory and global optimisation. Arrays of these elements are variously known as Neural Networks or Connectionist Machines and collected under the term Parallel Distributed Processing (PDP), [1], [2].

In order to explain the connection between biology and electronics, a simple overview is given of a biological neuron together with a discussion of the key aspects which should be incorporated into a neural model. The extension

of the model to a network capable of collective computational behaviour is described and the necessity of requiring 'hidden elements', not directly connected to the environment, but nevertheless influencing the behaviour of the network so that it can solve 'interesting' problems is explained. The central key problem at the heart of all neural network research is 'Credit Assignment' which requires a method for efficiently adapting the connection strengths or 'weights' of these hidden elements so that the network operates correctly. Various approaches to this problem are utilised by the different neural network architectures.

Two main classifications of these arrays exist in the literature. The first are Constraint Satisfaction Networks such as Hopfield networks, [71], and Boltzmann machines, [72], [73] which use energy minimisation gradient descent techniques to carry out global optimisation. The second are Multilayer Back-propagation Networks which utilise deterministic elements and error gradient descent to minimise the difference between actual and desired response, [74].

The chapter briefly reviews optimisation by simulated annealing and describes the travelling salesman problem, [71], [75], which has practical importance in areas such as optimum routing and placement in VLSI circuitry. It is explained how such an approach may be adapted for use in these Constraint Satisfaction Networks which use energy minimisation via supervised learning (or learning with a teacher). The chapter then explains how the Multilayer Backpropagation network gradient descent technique uses a second reverse-pass through deterministic elements to reduce an error cost function using supervised learning. Previous applications of this method have allowed

successful Text to Speech conversion, using training by example, [76], [77]. References [78], [79], [80] and [81] have also been found useful as background reading.

During the course of this work, a versatile neural network simulation program has been developed (documented in Appendix Four), and example output from a trained network performing simple character recognition is presented. Whilst neural networks may be taught to recognise an input and produce a required output, it is possible to argue that instead, all the input–output options could be stored as truth tables in cheap RAM memory, [82]. This approach is satisfactory where the environment is known, however, this research is predominantly concerned with interaction with a random environment where the responses are less well defined and the network must be able to optimise its behaviour in this situation.

One of the strengths of neural networks lies in their potential ability to generalise, particularly where there is natural redundancy in the information. However, extreme care must be taken to ensure that the network has sufficient information available from which to produce the correct generalisation. Ambiguous or ill–presented information is likely to result in disappointment.

5.2 Neural Models and Networks

This section gives a basic description of the biological neuron and extracts the key features which are then incorporated into rules and conditions used to describe the operation of a computer model. The neurons form the lines of communication used by the brain to process complex information [83].

The electrical signals may be recorded using intra or extra-cellular recording techniques, [84]. The information content of the signals is encoded, not wholly in the signal itself but in the specific interconnections of the neurons, whereby each neuron influences to some extent, all other neurons to which it is directly or indirectly connected.

By direct examination therefore, the structure of the brain is extremely difficult to analyse, [85]. Certainly, there may be degrees of specialisation in some areas of the brain, but more generally, knowledge and experience appear diffused across the neurons with the pattern of connections encoding the information, [10]. The analogy of a collection of interconnected electronic circuits such as a computer is often used to present the problem. It is not just the components which matter but also their specific interconnection which describes the underlying principle of operation. The brain is several orders of magnitude more complicated than current computers, possessing 10^{10} to 10^{12} 'components'. Different neurons perform different tasks and this specificity appears linked to their chemistry.

The shape of the neuron also gives an indication as to its function. The arborisation gives an idea of the number of connections. The output of a neuron occurs through a long trunk known as the axon. The connections to other neurons branch off from this and the actual junctions are termed synapses. A single neuron can have up to two hundred thousand synapses, [10]. The impulse or signal is also known as an Action Potential. A typical signal from a neuron has an amplitude of the order of a few mV, lasting for about 1 ms and propagated via ionic action at up to 120 m/s. The

neurons fire pulses, the frequency of firing is dependent on the intensity of the stimulus. Examples of typical neuron output traces are shown in Figure 5.1. Figure 5.2 shows a typical pair of neurons, A and B. The presynaptic neuron A can influence the postsynaptic neuron B by either Excitation (which produces impulses in B) or Inhibition (which acts to prevent impulses arising in B). Neurons can receive many inputs from other neurons, this is known as Convergence and in turn, output to a great many others, known as Divergence as shown in Figure 5.3.

The process of summing together all the incoming excitatory and inhibitory inputs is termed Integration. The negative membrane potential undergoes potential reduction or Depolarisation up to a threshold at which point a pulse is triggered. The presynaptic neuron releases a chemical Transmitter substance in fixed degrees or 'quanta'. in response to a depolarisation which, at an Excitatory synapse, will encourage depolarisation in the postsynaptic neuron. At an Inhibitory synapse, the transmitter will tend to keep the neuron potential below threshold and prevent firing. Excitatory or Inhibitory action depends on both the type of transmitter from the pre-synaptic neuron and also the chemical receptors at the postsynaptic neuron.

After the firing of an impulse, there is a short Refractory period during which the neuron recovers. A pulse cannot be fired during this time. The neuron will continue firing at a frequency approximately proportional to the neuron potential subject to the above limits. Generally, neuron impulses pass 'one way' along synapses due to the ionic gradient.

5.2.1 Electronic Neuron Model

It is possible to infer some computational guidelines from these observations. Firstly, the information is preserved in the connections between neurons. If the inputs to a neuron are weighted (transmitter) and summed (membrane potential), then if the potential is above a noisy threshold, the neuron will fire. The frequency of firing is proportional to the size of the potential. This method may be considered as a form of stochastic coding, as introduced in Chapter Two, which is both a noise and fault tolerant, efficient way of enabling groups of single 'binary' channels, or synapses to carry and process information. The information cannot be carried to high numerical accuracy, but this not usually a requirement in biological processing.

The structure of this model is depicted in Figure 5.4 and shows the element receiving inputs which are restricted to the range between or equal to 0 and 1. In general, the work presented will be restricted to the binary values 0 and 1 although an example of one artificial neural network will be described which utilises real values between or equal to these limits. In addition to these inputs, it is usual to include an extra weighted input which is always of value 1 and known as the threshold input (sometimes referred to as the stimulus input). The weight on this connection is treated in exactly the same way as the others.

The total input at time t, s(t) to the neural element is a linear function of the inputs $x_n(t)$ and the weights, $w_n(t)$ on these connections.

$$s(t) = \sum_{n=1}^{N} x_n(t) w_n(t) \tag{5.1}$$

(This expression should be compared with the similar expressions evaluated in Chapters Two and Three). It may be seen that the action of the threshold input is to ensure that in the absence of any other inputs, s(t) is never zero. The most basic element could be considered to switch on for +ve s(t) and off for −ve s(t) as represented by the step function shown in Figure 5.5a. This model is improved by making probability of the element firing at time t, proportional to this total input with upper and lower bounds as shown in Figure 5.5b. This function may be represented smoothly by adopting the function;

$$p(t) = \frac{1}{(1 + e^{-s(t)/T})} \tag{5.2}$$

This function is easily differentiable and saturates at 1 for large positive s(t), and 0 for large negative s(t). The T term may be used to adjust the gradient and as T approaches zero, the function more closely approximates the step function of Figure 5.5a. Some sources choose to call the T term 'temperature' and this is explained later. This function is plotted in Figure 5.5c for three different values of T, 0.25, 0.5 and 1.0. Stochastic encoding may be performed by generating the p(t) as shown in equation 5.2 and then thresholding against a random noise threshold such that the output of the element y(t) equals 1 with probability p(t), otherwise y(t) equals 0.

The response of the model clearly lies in the values of the input weights which may be positive (encouraging firing) or negative (discouraging firing). The major problem is adapting these connection weights between neurons to enable them to correctly process and store information, [86]. The mechanisms used in real biological neurons are not well understood, however

two main approaches have been suggested for use in artificial neural networks. The first is generally taken to be a variant of a proposal known as the Hebbian Rule, [87], which says that Pre–Synaptic and Post–Synaptic activity increases pathway efficacy, ie;

$$w_n(t+1) = w_n(t) + cx_n(t).y(t) \tag{5.3}$$

$w_n(t)$ represents the weight or connection strength between two neurons, $x_n(t)$ is one of the n inputs to the neuron and y(t) is the output of the neuron. Equation 5.3 simply states that, examining each input to the neuron in turn, if the output y(t) of the neuron is equal to its input $x_n(t)$ then the weight corresponding to that input, $w_n(t)$ should be increased by a small positive constant c.

The second approach is an error–reducing technique known as the Delta Rule, [88]. The Delta Rule requires that the system is presented with pairs of input–output vectors. The system uses the input vector to produce an output vector and then changes the weights to reduce the difference between the desired and the actual output vector, effectively performing gradient descent of an error surface. This is discussed later in the section describing error backpropagation networks.

5.3 Generalisation and Discrimination

In the introduction to this chapter, it was mentioned how neural networks have the potential ability to generalise from partial or incomplete information. The requirement for so–called 'hidden elements' was also stated,

for the solution of 'non-trivial' problems. It is necessary to examine the operation of the electronic model as shown in Figure 5.4 in more detail to explain these statements.

5.3.1 Linear Discriminant Functions

Consider the first part of the model which forms the weighted summation. An input vector X, is multiplied by the weight vector W resulting in a scalar product S. Consider the case where the element uses a binary threshold, where, if the product S is positive the element outputs a one, otherwise it outputs a zero. The action of the W vector is to separate the clusters of data mapped by sets of input X vectors with a single decision surface or hyperplane vector. This hyperplane vector is known as a Linear Discriminant Function in an N-dimensional space and may be explained as follows, [89], [90].

Consider the simple logic gate truth table shown in Table 5.1. The neural element, through repeated and random presentation of the four, two dimensional X vectors, which are the inputs in the table, seeks to reproduce the correct outputs in response to these inputs, through some suitable algorithm which adjusts the W vector. For clarity, this explanation will neglect the threshold input although extension to this case is trivial. The X vector input to the element may be considered to correspond to a simple set of patterns, with the element required to 'recognise' or respond in some way to the case when only one particular binary 'pixel' input is on, as shown in Figure 5.6. The X vector inputs may be mapped onto a unit square and the required

outputs drawn at these points as shown in Figure 5.7.

Consider the weight vector W and the input vector X at time t. The output of the element is determined by the value of the internal weights $w_n(t)$ and the input vector $x_n(t)$. Recalling equation 5.1;

$$s(t) = \sum_{n=1}^{N} x_n(t).w_n(t) \tag{5.4}$$

The weighted sum s(t) is the inner or scalar product of the x and w vectors, ie;

$$\begin{aligned} s(t) &= \overline{W}.\overline{X} \\ &= |W|.|X|cos\theta \end{aligned} \tag{5.5}$$

where θ is the angle between $\overline{W}$ and $\overline{X}$. A positive product occurs when $\theta < 90°$, in other words, for x coordinates below the decision surface, which may be considered as a line drawn perpendicular to the weight vector. A negative product occurs when $\theta > 90°$, ie. for x coordinates above the decision surface.

It may be seen that the element is operating by forming a Linear Discriminant function. Therefore if the element successfully separates the region of the unit square into an area where it should be 'on' and another where it should be 'off' by adapting the weights then it will have solved the problem. Furthermore, if, for instance, non-binary values for the x inputs were used which had not previously been presented, then the same weights would enable the element to decide whether it should be on or off. In effect it would have generalised from the original training set but this generalisation need not necessarily be the correct one.

5.3.2 Hidden Elements

Consider a more complex case, the Exclusive OR operation. This is the simplest case of an even parity check and the familiar truth table is given in Table 5.2. If the input vector is mapped as coordinates on a Unit Square, as shown in Figure 5.8a, then it is clear that no single decision line exists that will separate the two classes of data corresponding to the correct outputs. Therefore a single layer network, in this case, a single element will be unable to effect the correct transformation of the input vector to the output vector (note the special case where the input and output vector are presented together as one single vector, these are solvable by single layer networks).

For this reason, the Exclusive OR is referred to as a non-trivial problem which can be shown to require a second 'Hidden' layer in the network to solve the problem. This extra layer enables the network to develop its own internal representations as follows. By using a second element connected to the output of the first, as shown in Figure 5.8b, then assuming the first element forms the linear discriminant vector as shown in Figure 5.8c, the second element can then 'decode' the output of the first element plus the original input vectors.

Therefore (excluding the threshold input), the second element has three inputs so that, for the second element, the Unit Square becomes a Unit Cube and the classes are now separable by a two dimensional decision surface enabling solution of the problem as shown in Figure 5.8d. It may be shown that when many elements and layers are involved, the large-dimensional

space created enables solution of quite sophisticated discriminant problems, [5], although generally speaking, the larger the dimensionality, the longer the learning time required.

5.4 Credit Assignment

The training of the networks, such that they produce the correct output in response to their inputs, requires that the weights are adjusted in a suitable way to enable a discriminant function to successfully separate the clusters of data in N-Dimensional space. The networks are required to be self-organising and must adjust these weights without any human intervention. Clearly some method of detecting the error involved in the values of the weights for each element is necessary.

For single layer networks, whose elements are directly connected to the output, observation and hence weight adjustment would be easy, but these nets cannot cope with non-trivial problems. The problem of assigning error to the hidden elements of a multi-level network is known as 'Credit Assignment'. It sounds deceptively simple but led to the demise of the early networks such as the 'Perceptron', [91], [92], which could only solve the single layer and therefore trivial problems. The neural networks to be described each have their own method of adjusting these 'Hidden' weights.

5.5 Network Topology

The neural model as depicted in Figure 5.4 needs to be connected in a network of such elements to be of any practical use. The network needs

to be able to access information from outside the network in the form of an input vector and return information back in the form of an output vector.

Clearly there are a great many potential interconnection combinations for N neurons with M inputs per neuron in an electronic network. This thesis concentrates on two possible combinations from this set. The first is a totally interconnected recursive network as shown in Figure 5.9. It may be seen that each element output forms an input to the other elements although there is no connection from an element to its own input. The second case is that of a multilevel feedforward network as shown in Figure 5.10. In this case, elements in the first level receive inputs and their outputs form inputs to the next level and so on. The output of the array is taken from the final level.

Whilst many other topologies are possible, these two cases have been widely explored in the literature and cover the most important features. It should be noted however, that the special properties of each of these two networks will be shown to be dependent on the structure and interconnection of the individual neurons and this suggests that specific applications are likely to develop their own specialised interconnection arrangements. One area where neural networks have been shown to have considerable success is in global optimisation.

5.6 Optimisation

There are many Engineering problems which require the optimisation of a set of parameters affecting some complex system, [20], [62]. This usually

involves the minimisation of a cost function representing a quantitive measure of the 'performance' of the system in order to locate the solution at the global minimum. Since this procedure may require the adjustment of many parameters, an exhaustive search for an exact solution, through all the possible combinations of parameters becomes rapidly unproductive due what is known as 'combinatorial explosion'.

For instance, if the number of possible options for solution of a problem requires a complete search of N permutations, then a complete search consists of N! (N factorial) potential solutions. This class of problem, for which an exact solution is very difficult to determine in this way is known as NP–complete, where NP is Nondeterministic Polynomial time, since the number of steps required to arrive at a solution grows faster than any finite power of the size of the problem.

In other words, simply increasing the power of the computer gives little corresponding benefit in terms of decreasing the time taken to exactly solve the problem. For these, a polynomial time exact solution may not exist but there may be algorithms which offer good approximation in polynomial time.

5.6.1 Travelling Salesman Problem

One NP–complete problem which is often quoted in the literature, is known as the Travelling Salesman Problem (TSP), [71], [75]. The problem consists of a salesman who is required to visit a number of cities. The salesman is constrained in that he must visit each city only once, and

complete his journey at the initial starting city. Additionally, he must take the shortest route. Therefore, the cost function to be minimised is the total distance covered such that the shortest route between the N successive cities is chosen.

Figure 5.11 gives an example of the potential routes for a four city network commencing and concluding at the first city. Clearly, a thorough computer search would need to examine all the alternatives, the number of which grow rapidly with increasing N and is therefore impractical. However, note that in this example, the best route need not necessarily be chosen as a reasonable route could be considered acceptable. In this case, a local minimum of the distance cost function which lies close to the global minimum will be acceptable. The TSP problem involves minimisation of a distance function which requires non-violation of a number of constraining conditions. The technique used to solve the problem is therefore extremely relevant to routing problems such as optimum component placement, PCB tracking and VLSI area efficient design.

Work in theoretical physics has suggested forms of neural network which perform optimisation by locating acceptable minima using equilibrium methods which have their origins in statistical mechanics. These networks are known as Constraint Satisfaction networks since they seek the optimum solution subject to the constraints of the problem. Two examples of this approach utilise forms of the recursive connection topology mentioned previously and are known as the Hopfield Network and the Boltzmann machine. In order to understand this approach, it is first necessary to briefly review work in

optimisation which took place at roughly the same time using statistical mechanics. This was known as Optimisation by Simulated Annealing or OSA.

5.6.2 Optimisation by Simulated Annealing

A method of performing combinatorial minimisation by using techniques originating from statistical mechanics was reported by Kirkpatrick et al, [93]. The essence of this work was as follows. Consider an optimisation problem which requires a cost function $f(X) = f(x_1, x_2, ..., x_N)$ to be minimised and a set of potential solutions generated by trial moves. Figure 5.12 illustrates an example of such a typical cost function. Note the two closely spaced minima near the global minimum, each representing a good solution to the problem. This function could for instance represent distance travelled, as in the Travelling Salesman problem, in which case, the trial moves would consist of a potential route fulfilling all the constraints.

Normal gradient descent methods would evaluate f(X) and accept the new set of moves with probability one if f(X) was less than the previous value of f(X). This has the disadvantage that local minima may trap the process, representing a suboptimal solution. Simulated annealing however, allows the acceptance of moves which increase f(X) with a probability dependent on a so-called 'temperature' term which is decreased very gradually as the process continues. This allows the system to escape from local minima as long as the 'temperature' is reduced slowly in a series of time steps known as an 'annealing schedule'.

Simulated annealing has its roots in statistical mechanics which seeks to describe the global properties of large numbers of atoms in liquid or solid matter samples, [94]. Due to the large numbers of atoms, only the most probable behaviour of the system in thermal equilibrium at a given temperature is observable. The probability of each atom being in a particular energy state may be described by a function known as the Boltzmann distribution, which is temperature dependent, [94], [95]. In this way, the interaction of each atom with its neighbours and also the total energy of the system can be described. The way the system behaves as a whole may be examined, particularly at the low temperatures. Ground states and configurations close to them in energy are extremely rare among all the possible configurations of the global body and yet they dominate its properties at low temperatures since as T is lowered, the Boltzmann distribution collapses into the lowest energy state or states. Note though that it is not enough in practice to simply cool a substance in order to make it adopt the ground or lowest energy state. Consider for instance a crystal, this represents the globally minimum energy state for a substance but careful preparation is necessary to grow one involving a careful annealing process requiring that the substance is melted and then slowly cooled with a long time spent at the lower temperatures. Faster cooling can result in the system not achieving equilibrium at each temperature and resulting in a crystal containing many defects. In the worst case, the substance may form a glass, with no crystalline order at all. Such a state consists of only locally optimal energy states.

It may be seen that this process is analogous to optimisation, where one seeks the lowest energy or ground state of the system. The atoms of the crystal have no knowledge of their part in the substance as a whole, only local interactions take place, yet by this process global optimisation may take place. If then, an optimisation problem can be expressed in terms of some 'energy' of a 'similar' system, methods based on this approach could allow global optimisation using only local updating. In a real problem, gradient descent methods which only accept rearrangements (or new moves) which lower the cost function of the system from the previous value, may be compared to rapid cooling from a high temperature to T = 0 giving resulting solutions which are likely to be sub-optimal. Simulated annealing offers two useful features. The first is that the procedure does not necessarily get stuck at local minima, since at high temperatures, the system may escape from local minima. The second is that gross features appear at high temperatures, whereas finer features appear at low temperatures. This enables a natural, gentle division of the problem into clusters. The relevance of such methods of optimisation to neural nets follows naturally from this work.

5.7 Constraint Satisfaction Networks

It was explained in the previous section how equilibrium methods may be applied to optimisation problems. This was extended to two neural networks as follows.

5.7.1 Hopfield Networks

The Hopfield Network reawakened research activity in neural networks, [71], [75], [96], [97]. Apart from a few workers, interest had waned following the analysis of Perceptrons which showed the limitations of single layer networks, [92]. Hopfield described a network of simple recursively connected elements as shown in Figure 5.13, [71]. As described earlier, the output of each element forms an input connection to the other elements. The operation of the basic element is exactly as described in section 5.2 and the output function of the element is the step function or binary threshold, obtained by setting T to zero. The element therefore has two states. $y_i = 0$ ('not firing') and $y_i = 1$ ('firing') and is connected to the outputs of each of its neighbours.

Each of the connections has an associated weight which may be +ve 'excitory' or −ve 'inhibitory'. A special requirement for equilibrium is that the weights on connections between elements are symmetrical. This means, if element i is connected to element j by a connection of weight w_{ij} then element j is connected to element i by the same weight. ie.

$$w_{ij} = w_{ji} \tag{5.6}$$

The output of an element does not appear at its own input. ie.

$$w_{ii} = 0 \tag{5.7}$$

The weight update is a variation of the Hebbian Rule, [87], and requires that the weights are updated asynchronously with the incremental change in weight, Δw_{ij} computed as;

$$\Delta w_{ij} = [2y_i - 1][2y_j - 1] \tag{5.8}$$

This may be interpreted as saying; if element i has the same state as element j, then increase the link between them by +1, otherwise decrease it by -1. This therefore increases or decreases the link between the two items or 'hypotheses' represented by the two neurons. In this way, each neuron element has a degree of influence over all the other elements in the network. If the elements are randomly updated asynchronously according to this threshold rule, then as long as the weighted connections are symmetrical, the updates will eventually stop as the network converges to a stable state.

For this class of network, an arbitrary function may be defined which relates the inconsistency between the weights on connections to each element, to the state of that element. In the literature, this function has been termed the 'energy' of the system since it is analogous to spin interactions in systems of magnetic atoms. The stable state represents a minimum of the energy function, not necessarily the global minimum representing a solution satisfying all the constraints. This energy function is given by, [80],

$$E = -\sum_{i \neq j} \sum w_{ij} y_i y_j \tag{5.9}$$

The equation may be explained as follows. If many pairs of active elements are on, which are joined by inhibitory weights, then the network is highly inconsistent and the energy is high. Similarly, if they are joined by excitory weights, then the energy is low. The energy function then provides a measure of how well the constraints are being satisfied by the network.

Given equation 5.9, Hopfield shows that for this network, the energy of the system is always reduced on each update, the difference in energy of

the system when element i changes state from off to on (ie. y_i changes from 0 to 1), is therefore given by;

$$\Delta E_i = \sum_j w_{ij} y_j \tag{5.10}$$

Operation of the array occurs in two phases; 'clamped' and 'unclamped'. In the first, the training phase, the outputs are clamped, ie. fixed and the weight matrix is free to update. In the second, the executable phase, some or all of the outputs are free to vary (unclamped) but the weight matrix is fixed.

Network learning is performed by 'clamping' some or all of the neuron outputs to a set value irrespective of whether their weighted sum would require them to fire or not. In this way, the weights will adjust themselves to values which make this a stable position for the array. Once this training phase has been performed and the weights do not change, they are fixed. If some or all of the outputs are 'unclamped', the neuron outputs should faithfully recreate the entire output as it was during 'clamping'. We may now say that the array has 'learned' to produce the desired output although all that has been done is to use the weights to form a minimum in array energy space such that the system will always tend to form this output.

The array may be taught multiple vectors by randomly presenting each vector to the array in turn during training. If the array is subsequently presented with all or sufficient part of the vector, it will fall into the local minimum representing that vector. This Associative Memory function has been found to be strongly dependent on the Hamming Distance between

vectors, [71], and not as reliable or efficient as Matched Filter techniques, [98], although other groups have investigated alternative storage techniques, [99].

An example of a Hopfield Net with 5 x 5 elements learning to recognise the figure zero is depicted in Figure 5.14. Figure 5.15 shows the connection weights for each element. Each block represents an element and each square in the block represents a weight, red for excitatory and black for inhibitory. The magnitude of the weight is indicated by the area of the square where the maximum area is scaled to the maximum weight to be plotted in the diagram and is listed at the base of the figure. The position of the square in the block indicates to which element, that weight connects. Therefore, as a consequence of equation 5.7, there will be one blank square representing null connection from the element to itself. Weights are symmetric, eg. the connection of the top left element to the bottom right element, represented by the bottom right square in the top left block, is the same value as that shown by the top left square in the bottom right block. The small square at the top left of each block represents the threshold input. This enables the element to set a non–zero threshold. Examining the figure, the pattern of weights reflects the stored figure zero, recall of which requires that all the perimeter elements fire, but that all enclosed elements remain off.

As has been already stated, the Hopfield network can be made to converge to local minima which represent stored items in an Associative memory. However, when used to solve constraint satisfaction problems, the method of gradient descent of the energy surface can allow local minima can

trap the network before a global solution is reached. This problem was solved by a modification to the expression used for updating the element, allowing the network to perform simulated annealing as explained previously, and led to the development of a neural network known as the 'Boltzmann Machine'.

5.7.2. Boltzmann Machine

The Boltzmann machine, [72], [73], [100], forms a natural extension of a Hopfield network and incorporates simulated annealing and other terms from statistical mechanics in order to learn. Recall equation 5.10 which gives an expression for ΔE_i. In the Hopfield model, the element outputs a '1' or 'fires' if this value is positive with probability one. In effect, the element is rigidly following same decision path.

It was explained earlier, how simulated annealing enabled a probabilistic response to be incorporated into decision making and move acceptance. Consider equation 5.2 which describes the probability of output of the element. The Hopfield model operates using a binary threshold which may be obtained by setting the T term to zero. As explained, the T term controls the gradient of the probabilistic response. If T is decreased from an appropriate large value to zero, then the element response will initially be highly random, becoming more and more deterministic as T is reduced. This may be considered to be an annealing schedule and the T term is often referred to as 'temperature'.

Equation 5.2 then gives the probability that element i fires equal to P_i where;

$$P_i = \frac{1}{(1 + \exp(-\Delta E_i/T))} \tag{5.11}$$

This adjustment of the firing rule for the Hopfield element and a different updating rule for the weights resulted in a new network, the Boltzmann machine. This network had the additional advantage that it could incorporate hidden elements as shown in Figure 5.16, enabling it to solve non-trivial problems. The network still requires symmetrical connections to enable it to reach equilibrium.

The Boltzmann machine is normally trained to reproduce output vectors for given input vectors, by observing the probability distribution between each set of units, first when the visible elements are clamped by the Environment (state P^+) and then when running unclamped or totally free (state P^-), [100]. The object is to match the two probability distributions as closely as possible. This is performed by collecting statistics for all input vectors, at decreasing temperature intervals (ie. an annealing schedule), for each pair of connected units for the network at equilibrium. In this way, the connection weights may be adjusted by a factor proportional to $(P^+ - P^-)$. An additional absolute weight decay term is also introduced to prevent weights becoming too large. The necessity of collecting statistics over these temperature intervals results in a considerably slow training process which cannot necessarily be proposed as a suitable model of biological learning although by collecting the statistics, one is effectively using time averaging to extract information which is present in a stochastically encoded form. Alternative

distributions to the Boltzmann distribution have also been investigated, [101].

In summary then, Hopfield Networks and Boltzmann machines are a class of Neural Networks utilising symmetric connections, which can solve constraint satisfaction problems with varying ability and perform associative memory recall operations. The two methods share the technique of descent in network 'energy' space, the global minimum of which represents the optimum solution to the problem.

The Boltzmann machine avoids local minima traps by effectively introducing noise in slowly decreasing amounts into the system in a simulated 'annealing' approach. The whole process is analogous to a vast metal tray which has cup-like indentations at various depths, the deepest dent representing the overall solution to a problem. The tray has a solitary marble in it and one has to shake the tray, initially violently, then less and less, until the marble eventually falls into the deepest possible position.

This thesis does not consider this class of network further for the purpose of learning input–output vector associations as it is generally considered that they are inferior in terms of learning speed and computation, to a fairly recent and popular method known as Multi–Layer Error Backpropagation neural networks.

5.8 Multi–Layer Error Backpropagation Networks

Multi–Layer Error Backpropagation Networks, [74], use deterministic elements and back propagation of the error derivative to allow the network to encode input–output associations in a feedforward array. The one major

distinction of this type of network is that the operation of the element is deterministic. In other words, the the element operates as described in section 5.2 except that there is no stochastic coding involved and the output may take values between or equal to 0 and 1. Figure 5.17 shows an example of a multi-layer Backpropagation network.

Such networks have received perhaps the most notable acclaim for performing Text to Speech conversion based on learning by example, [76], [77]. The network in that case used many more hidden units than input or output units in order to store the large amount of information required. The weight updating follows the Delta Rule, [88], closely and the derivation proceeds as follows, [74].

5.8.1 Error Backpropagation

Consider the neural element shown in Figure 5.4. Let the element have an output y(t) which may lie between, or equal 0 and 1. The total input s_j to element j is a linear function of the outputs y_i of the elements which are connected from the previous level, to j and of the weights, w_{ij} on these connections together with the threshold input.

$$s_j = \sum_i y_i w_{ij} \tag{5.12}$$

The element output, y_j is a non-linear function of the total input.

$$y_j = \frac{1}{(1 + e^{-s_j/T})} \tag{5.13}$$

Note this is not a probability distribution as equation 5.2 and the T term

adjusts the gradient of this function as before. Define the total error, E, as;

$$E = \frac{1}{2} \sum_{c} \sum_{j} (y_{j,c} - d_{j,c})^2 \tag{5.14}$$

where c represents the input–output pairs, j represents the output elements, y is the actual state of an output element and d is its desired state. E is minimised using gradient descent and this method is a form of the Delta rule, [88]. The 1/2 constant term is introduced for convenient differentiation and in order to perform gradient descent, the partial derivative of E must be computed with respect to each weight in the network which is equal to the sum of the partial derivatives for each of the input–output cases.

Two passes are used, the forward pass is defined as when the elements in successive layers have their states determined by the input they receive from elements in previous layers. The backward pass propagates error derivatives from the final layer back to the first layer. The backward pass begins by computing $\frac{\delta E}{\delta y}$ for each of the output elements. Differentiating equation 5.14 for a particular case, c gives;

$$\frac{\delta E}{\delta y_j} = y_j - d_j \tag{5.15}$$

Using the chain rule to compute $\frac{\delta E}{\delta s_j}$

$$\frac{\delta E}{\delta s_j} = \frac{\delta E}{\delta y_j} \cdot \frac{\delta y_j}{\delta s_j} \tag{5.16}$$

Differentiating equation 5.13 gives;

$$\frac{\delta y_j}{\delta s_j} = \frac{1}{T} y_j (1 - y_j) \tag{5.17}$$

Substituting into equation 5.16 gives;

$$\frac{\delta E}{\delta s_j} = \frac{\delta E}{\delta y_j} \cdot \frac{1}{T} . y_j (1 - y_j) \tag{5.18}$$

Equation 5.18 shows how a change in the total input s to an output element affects the error. The total input is a linear function of the weights on the connections and so it is possible to compute how the error will be affected by changing these states and weights.

For a weight w_{ij} from i to j, the derivative is;

$$\begin{aligned} \frac{\delta E}{\delta w_{ij}} &= \frac{\delta E}{\delta s_j} \cdot \frac{\delta s_j}{\delta w_{ij}} \\ &= \frac{\delta E}{\delta s_j} . y_i \end{aligned} \tag{5.19}$$

and for the output of the i^{th} element, the contribution to $\frac{\delta E}{\delta y_i}$ resulting from the effect of i on j is;

$$\frac{\delta E}{\delta s_j} \cdot \frac{\delta s_j}{\delta y_i} = \frac{\delta E}{\delta s_j} . w_{ij} \tag{5.20}$$

By taking account of all of the connections diverging from element i;

$$\frac{\delta E}{\delta y_i} = \sum_j \frac{\delta E}{\delta s_j} . w_{ij} \tag{5.21}$$

Therefore $\frac{\delta E}{\delta y}$ may be computed for any element in the penultimate layer when given $\frac{\delta E}{\delta y}$ for any element in the last layer. This procedure may be repeated to compute this term for successively earlier layers, computing $\frac{\delta E}{\delta W}$ for the weights.

The important term is $\frac{\delta E}{\delta s_i}$ as shown in equation 5.18. Note how it is weighted and summed during the reverse error backpropagation pass as in equation 5.21 and compare this with the similar operation during the first

forward pass in equation 5.12. $\frac{\delta E}{\delta s_j}$ is also required locally to compute the error with respect to the weights as in equation 5.19. Whilst it is possible to use $\frac{\delta E}{\delta W}$ to change the weights after every input–output case, the method used in the majority of the simulations reported in this thesis is to accumulate $\frac{\delta E}{\delta W}$ locally over all the input–output cases before changing the weights. Gradient descent is then performed by changing each weight by an amount proportional to the accumulated $\frac{\delta E}{\delta W}$.

$$\Delta W = -\epsilon \frac{\delta E}{\delta W} \tag{5.22}$$

where ϵ is a small constant such as 0.1. An additional term may be introduced which takes account of previous weight changes to speed up the process where necessary.

$$\Delta W(t) = -\epsilon \frac{\delta E}{\delta W(t)} + \alpha \Delta W(t-1) \tag{5.23}$$

where t represents the current pass and α is a decay factor between 0 and 1 affecting the proportion of earlier gradients to the weight change.

In the simulations described in this thesis, this is usually set to an arbitrary value of 0.5. In the simulations, small random weights are introduced at the start of a simulation, however the method can still operate when starting from zero weights, as the logistic function, equation 5.13 has value 0.5 when the weighted sum is zero. Interesting weight plots may be obtained when using this learning method to reproduce figures. Figure 5.18 shows an example of a Backpropagation network of Length 5, Width 5 and Depth 2 learning to recognise and reproduce the figure zero as represented on a 5 x 5 matrix. Figures 5.19a and 5.19b plot the connection weights for

this case and it can be seen that the weight patterns reflect the nature of the problem.

5.8.2 Error Backpropagation element

It is possible to clarify considerably, the error backpropagation process by simply considering the process local to each element, in particular concentrating on the summation on both forward and reverse pass. Figures 5.20a and 5.20b show the Backpropagation element for both the final layer and previous layer neurons. In the forward pass, the final layer neuron provides an output Y_j formed from a function of the weighted sum of the inputs from the previous layer as previously described in section 5.2 but outputting a deterministic rather than stochastic value. On the reverse pass it receives the error from the Environment and computes the quantity $\frac{\delta E}{\delta s_j}$ which is passed up to the next level. It also locally computes the quantity $\frac{\delta E}{\delta w_{ij}}$ for each input connection given the output of the previous level present on each connection. This value is accumulated over the set of input patterns.

Neurons on the previous levels compute in the same manner but have the major distinction that the expression $\frac{\delta E}{\delta y_i}$ is now formed from the **weighted summation** of the input $\frac{\delta E}{\delta s_j}$ from all the elements in the next level, where the weights are those of the corresponding interlevel connections. The exponential function used in the forward pass to compute the output y_i of the element from the weighted sum is not used in the reverse pass to compute $\frac{\delta E}{\delta y_i}$ from the weighted sums of errors passed up through the array. A function of the output y_i is now used to compute $\frac{\delta E}{\delta s_i}$ for the element,

given $\frac{\delta E}{\delta y_i}$ and this is then output to the previous level and so on.

5.9 Conclusions and Summary – Chapter Five

This chapter has reviewed aspects of biological neural networks and incorporated these into a model which may form the basis of an electronic implementation. The chapter examined methods of performing optimisation using techniques from statistical mechanics. It was shown how these methods may be used in Constraint Satisfaction networks. Such networks are probabilistic in nature and their topology allows the use of equilibrium techniques to form local minima which can allow the network to perform associative memory or global optimisation.

The chapter concluded with a deterministic neural network which utilises a feedforward array topology enabling error feedback weight correction. Whilst this method is highly successful and popular, it is difficult to propose this method as a plausible biological mechanism since the element outputs are deterministic, contrasting with the pulse rate coding mechanisms observed in nature. A later chapter will present novel work which incorporates stochastic encoding techniques as described in Chapter Two with error backpropagation resulting in a novel network with considerable hardware simplification. Generally speaking, the majority of neural networks have been performed using serial computer simulation although the application of VLSI techniques is being considered, [102], [103]. The immense cross-connection required at the inputs of each neuron do present some problems for this technology and optical methods are also under consideration, [104].

This chapter concludes the first part of the thesis which introduced the basic concepts of stochastic encoding and described three simple two-state output stochastic elements. The first element, by forming the inner product function was shown to compute the Discrete Fourier Transform and Cyclic Correlation function when placed in an array of identical cells. It was described how the second element, operating as a stochastic automaton could learn to improve its performance and by operating in arrays, work as a team to perform optimisation and decision making. The third element, loosely based on the biological neuron was shown to learn input-output associations and perform optimisation. These elements are all from the same class of stochastic circuits which interact with a changing, possibly noisy, probabilistic environment and through improving their performance in some fashion, produce a collective computational response based on local operations.

The second part of the thesis considers hybrid approaches, combining stochastic techniques to form learning automata which perform pattern recognition in a neural network and examines a novel stochastic implementation of error backpropagation. The thesis then considers future work involving a unified neural element comprising the stochastic model described in section 5.2 with weight updating techniques taken from both error backpropagation and learning automata. This results in a simple distributed stochastic learning array offering simple hardware, flexibility and the capacity to interact with a changing probabilistic environment.

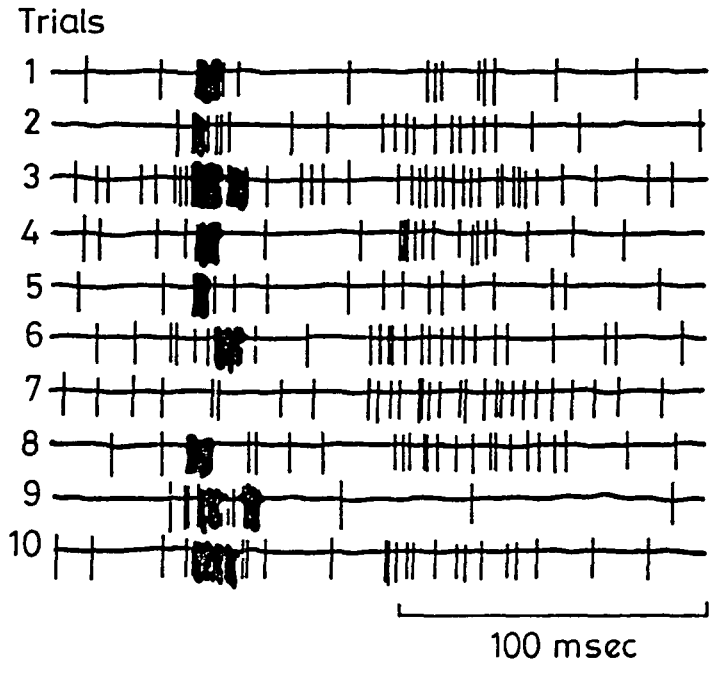


Figure 5.1 Typical neuron output traces

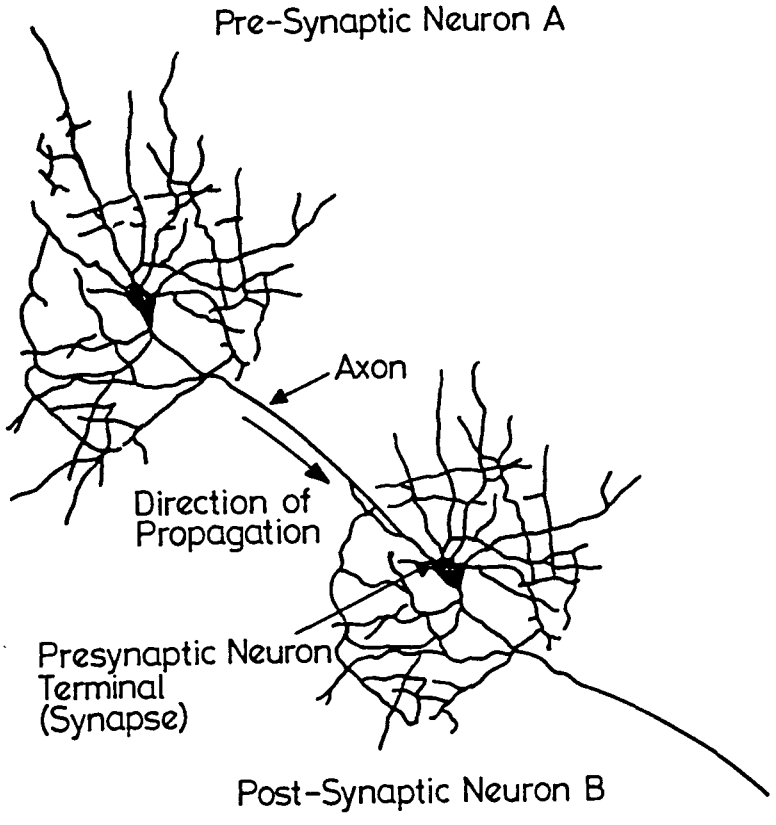


Figure 5.2 Pre- and Post-Synaptic Neurons

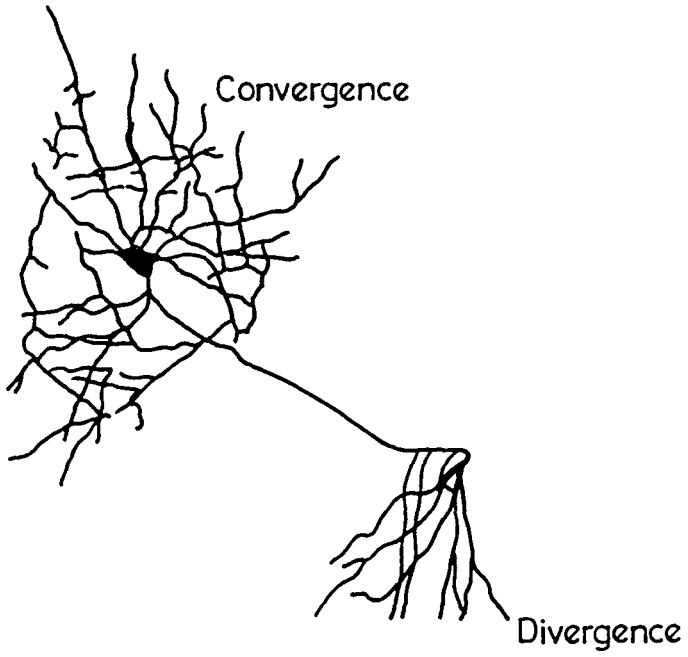


Figure 5.3 Convergence and Divergence

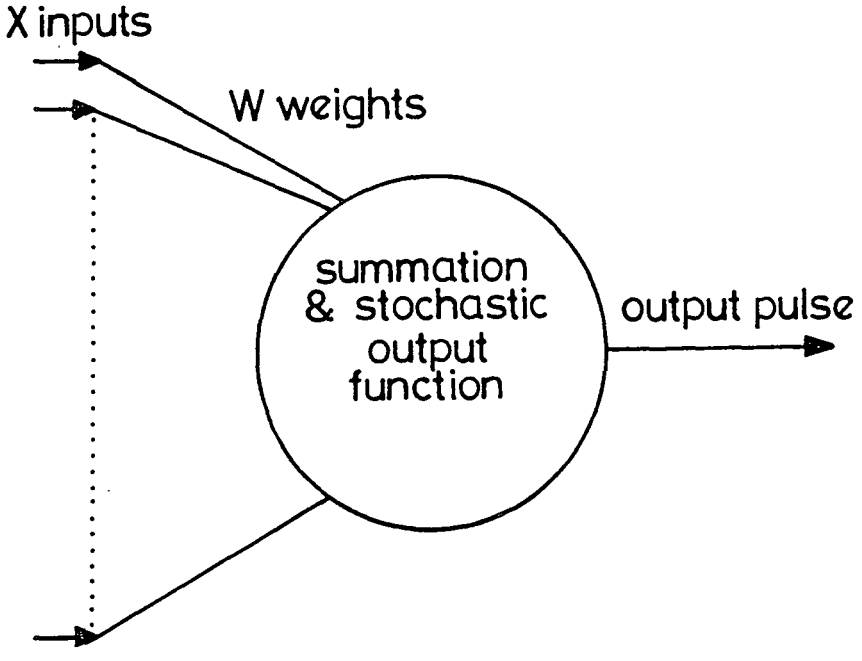


Figure 5.4 Neural Model

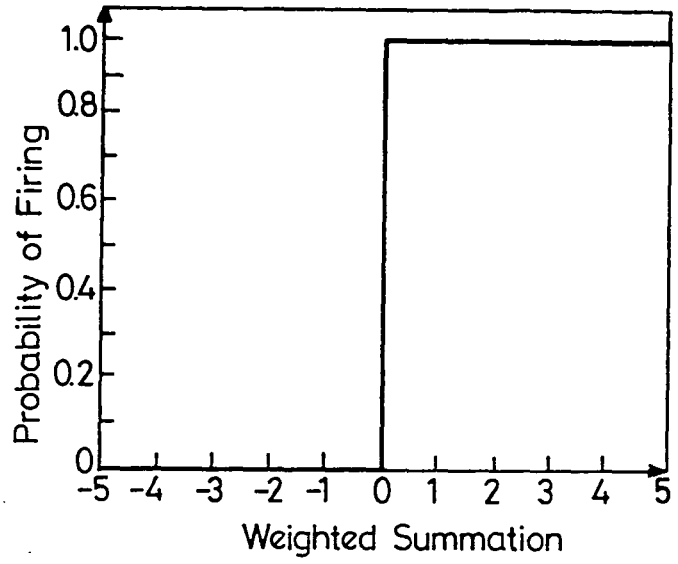


Figure 5.5a Step Function

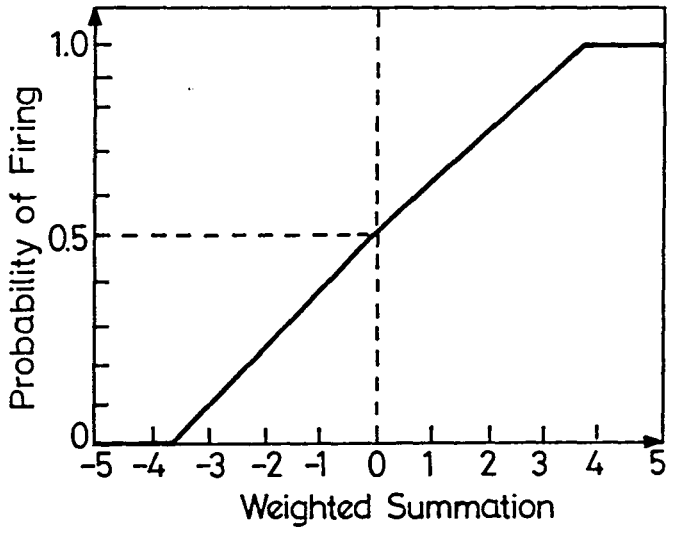


Figure 5.5b Proportional output function

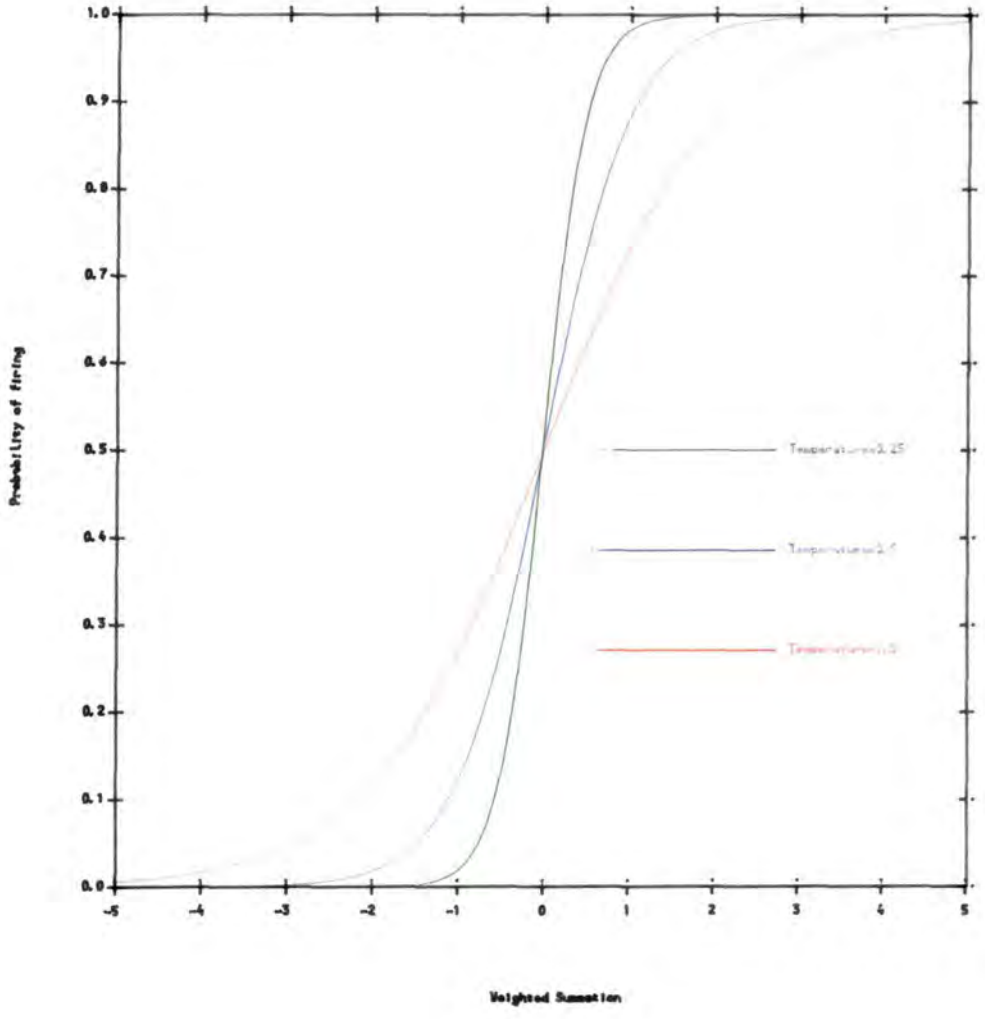


Figure 5.5c Probability of firing vs total weighted input

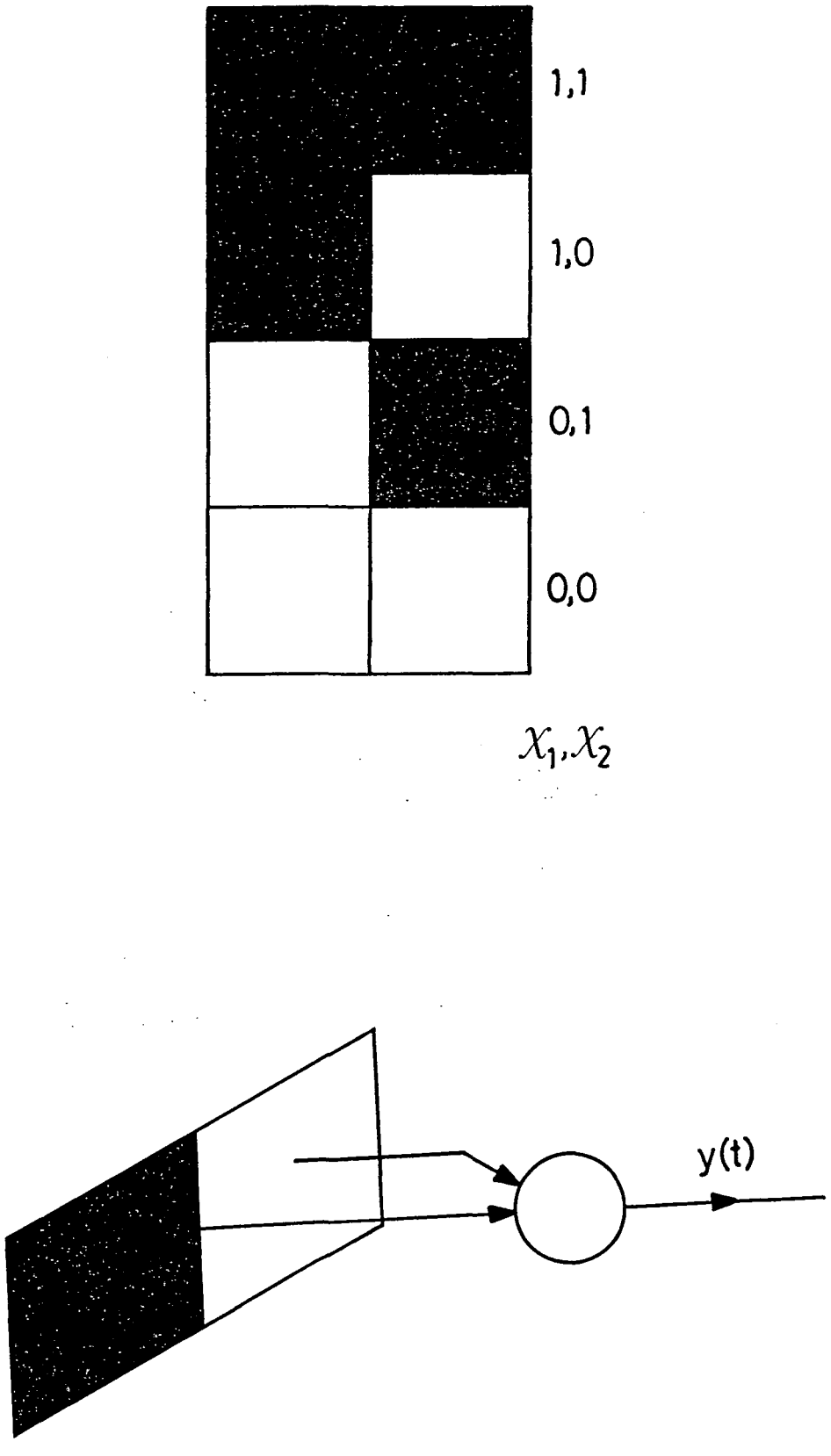


Figure 5.6 Simple pattern recognition with 2-bit input vectors

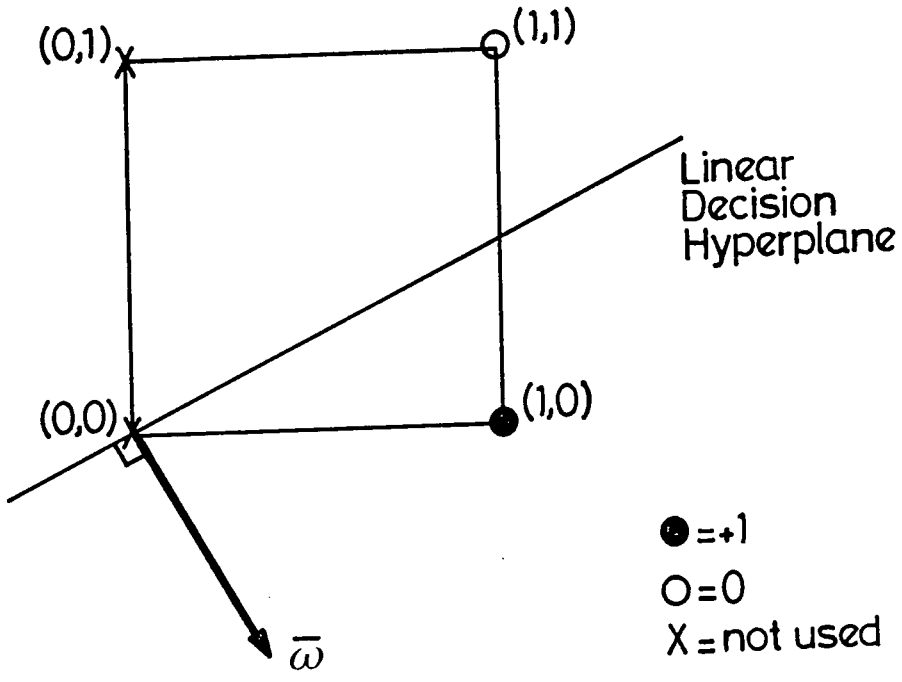


Figure 5.7 Unit square for Logic problem using one element

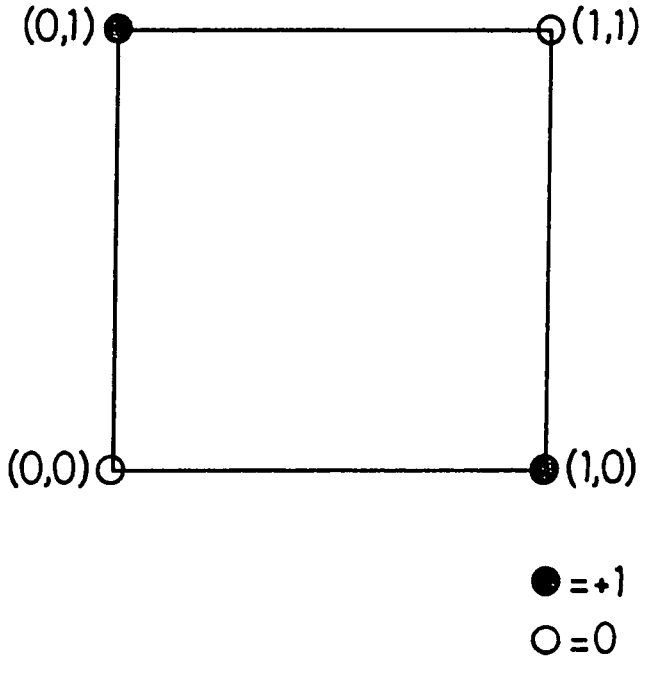


Figure 5.8a Unit Square for EXOR problem

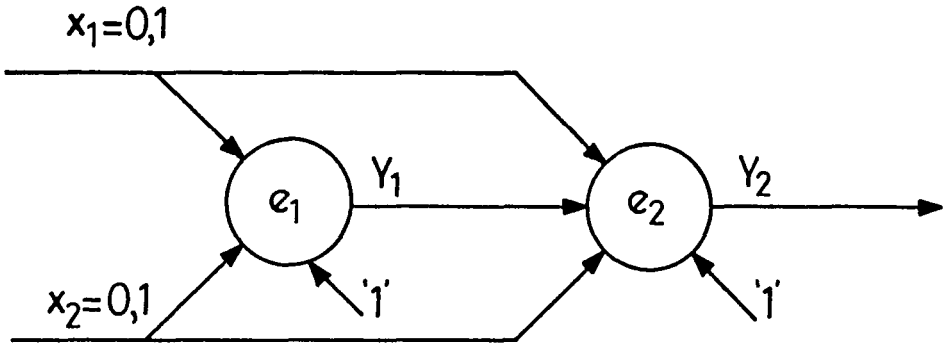


Figure 5.8b Exclusive OR with two Learning Elements

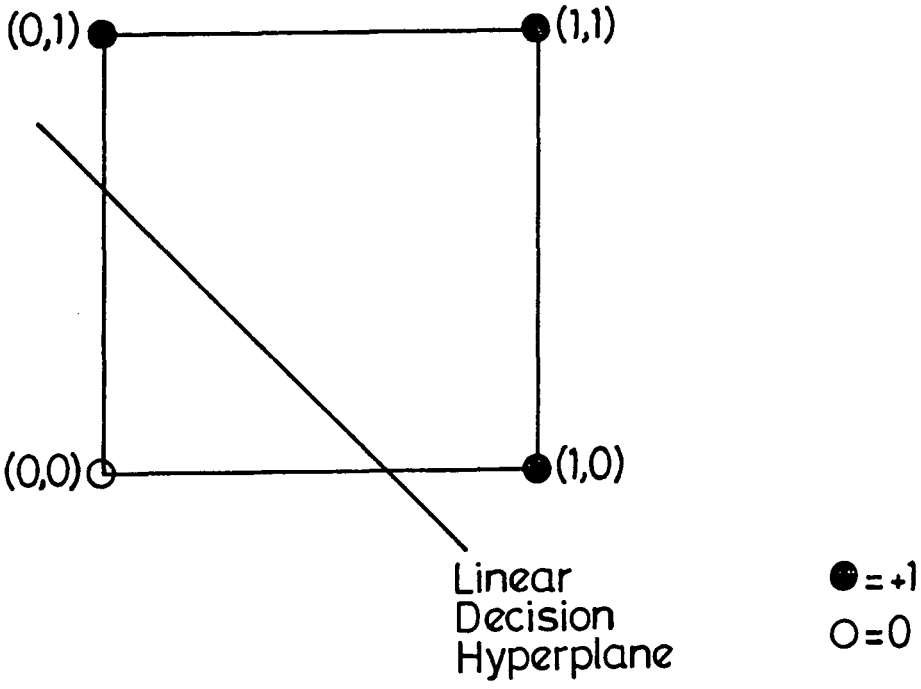


Figure 5.8c Unit Square for EXOR problem – Element 1

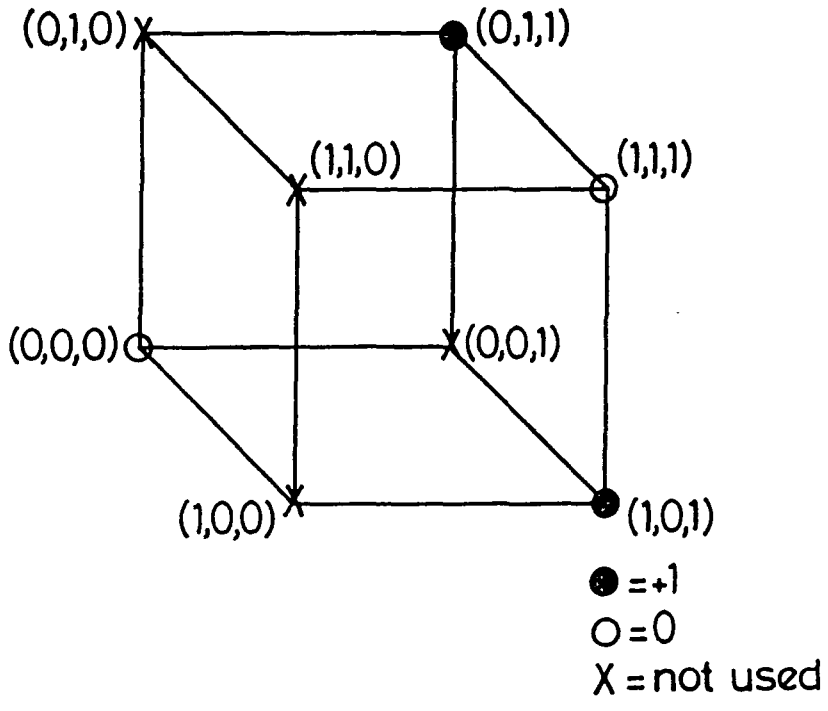


Figure 5.8d Unit Cube for EXOR problem – Element 2

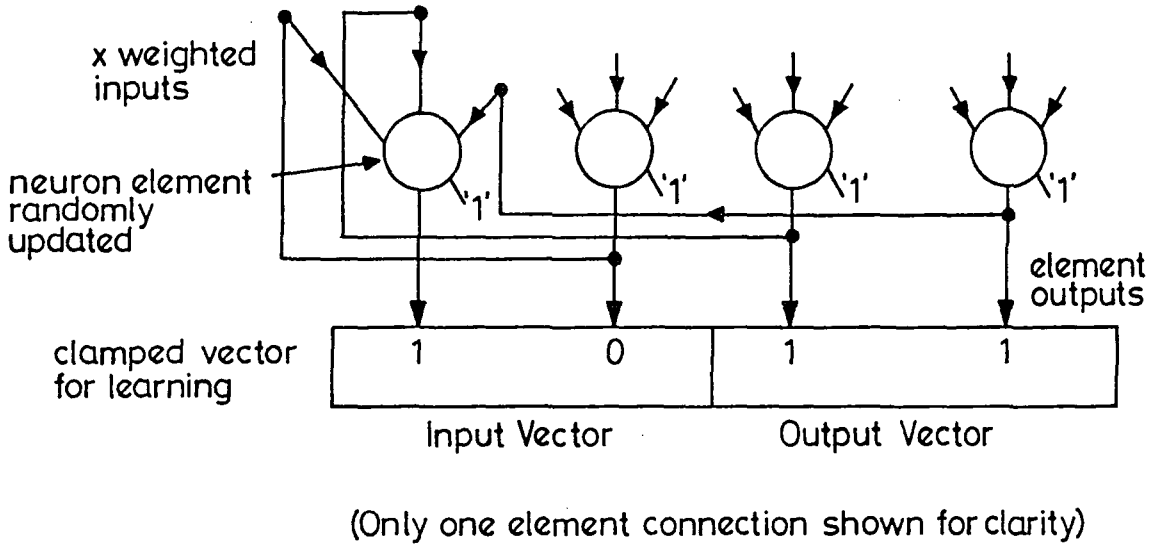


Figure 5.9 Recursively connected neural net

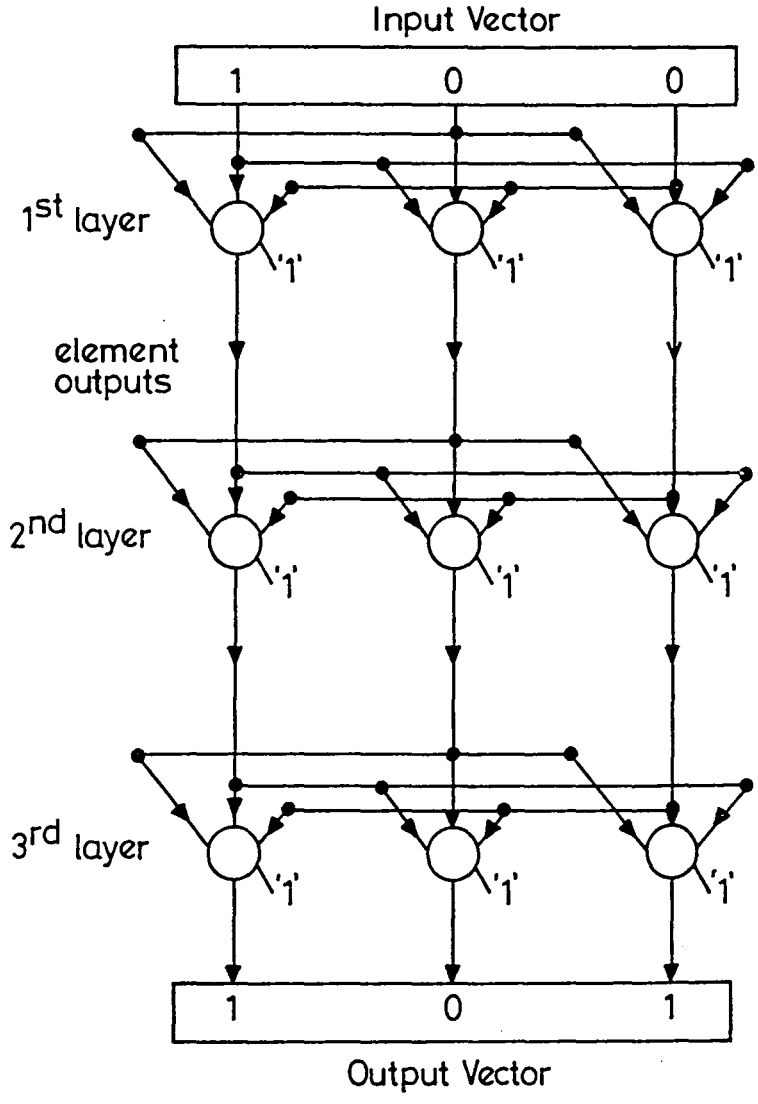


Figure 5.10 Feedforward neural net

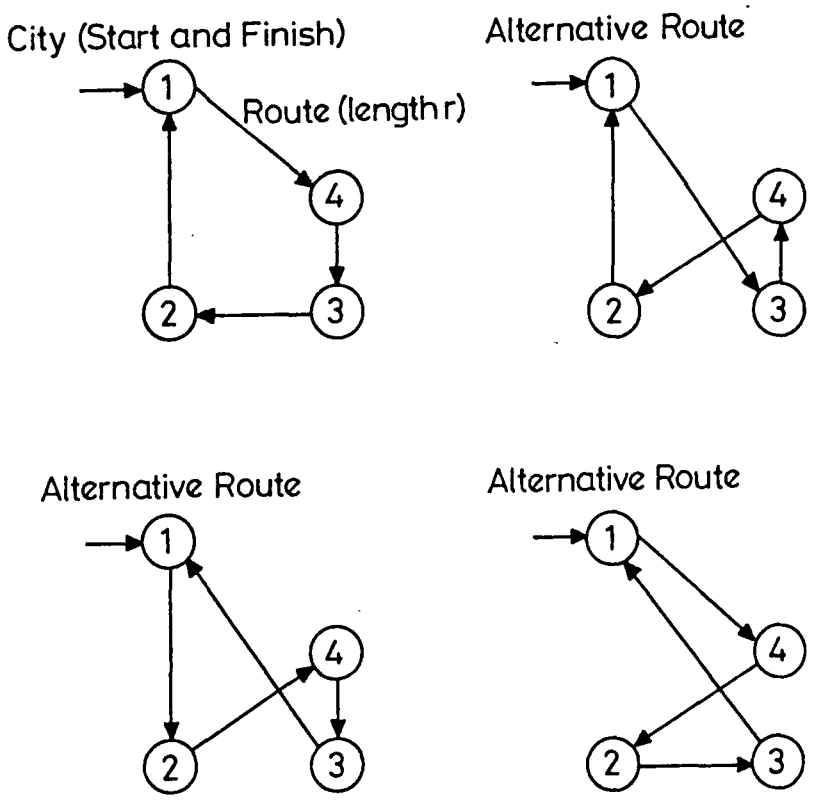


Figure 5.11 Travelling Salesman Problem

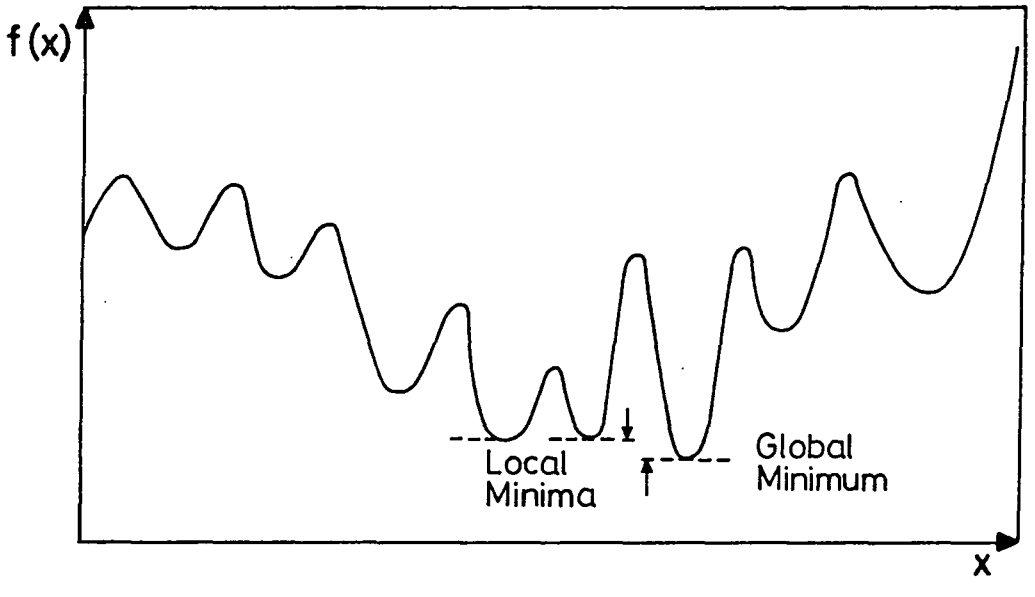
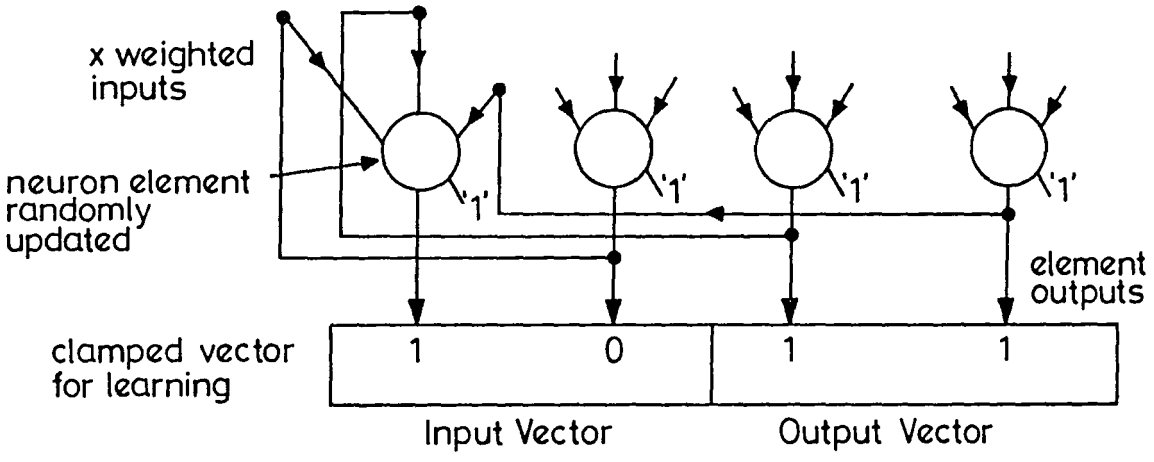


Figure 5.12 Cost Function containing Local and Global Minima



(Only one element connection shown for clarity)

Figure 5.13 Hopfield Network

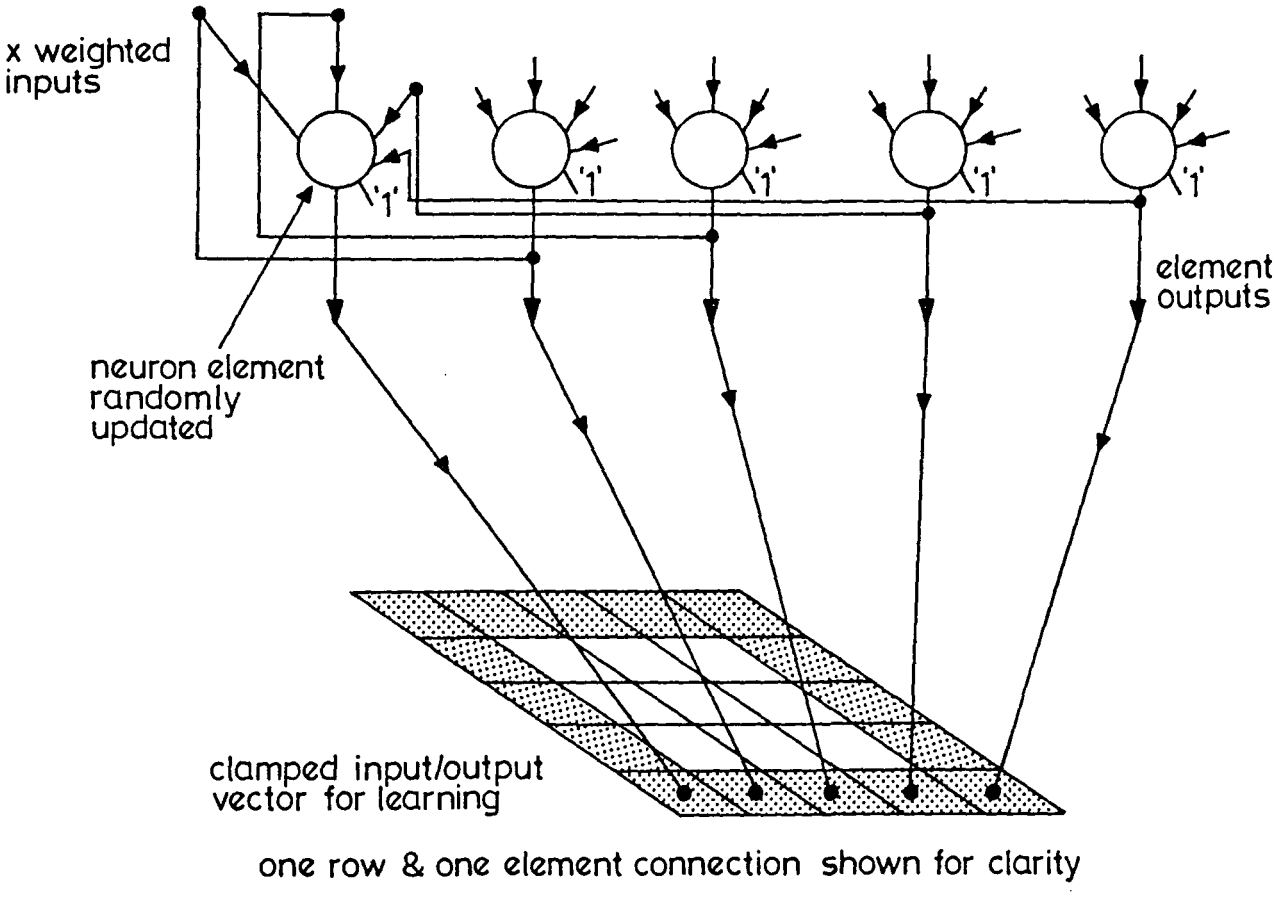


Figure 5.14 Hopfield Network learning to recognise a figure zero

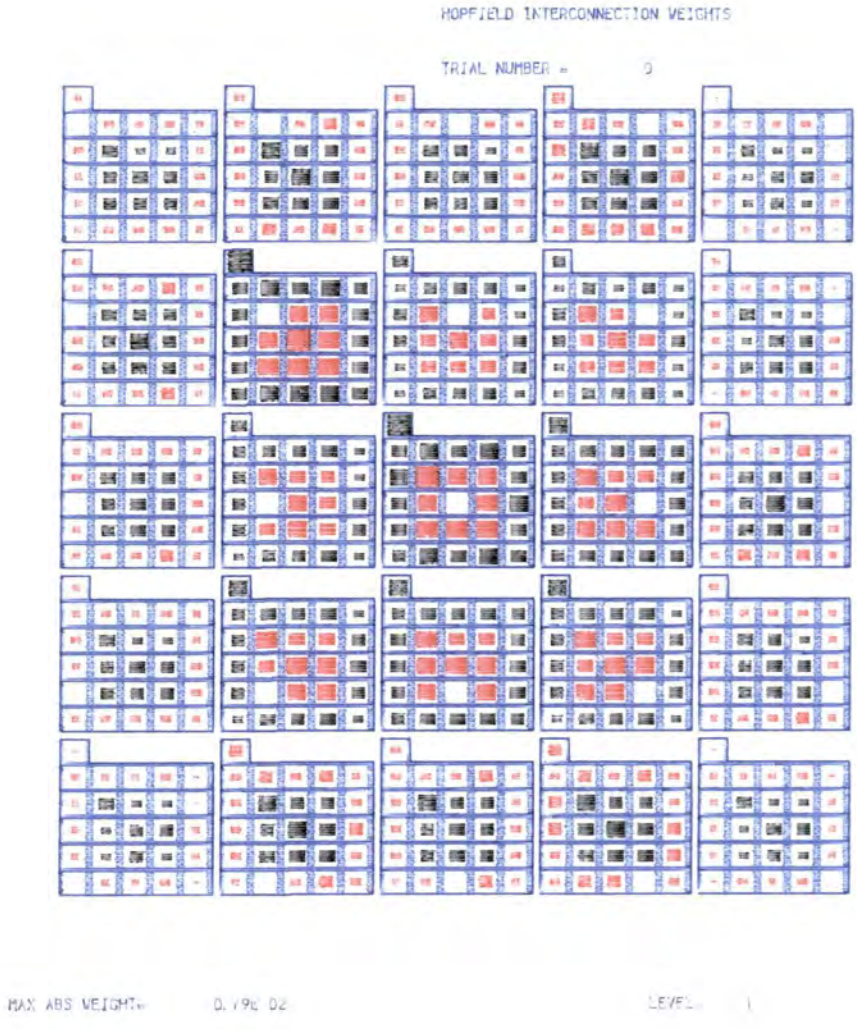


Figure 5.15 Hopfield Network – connection weights

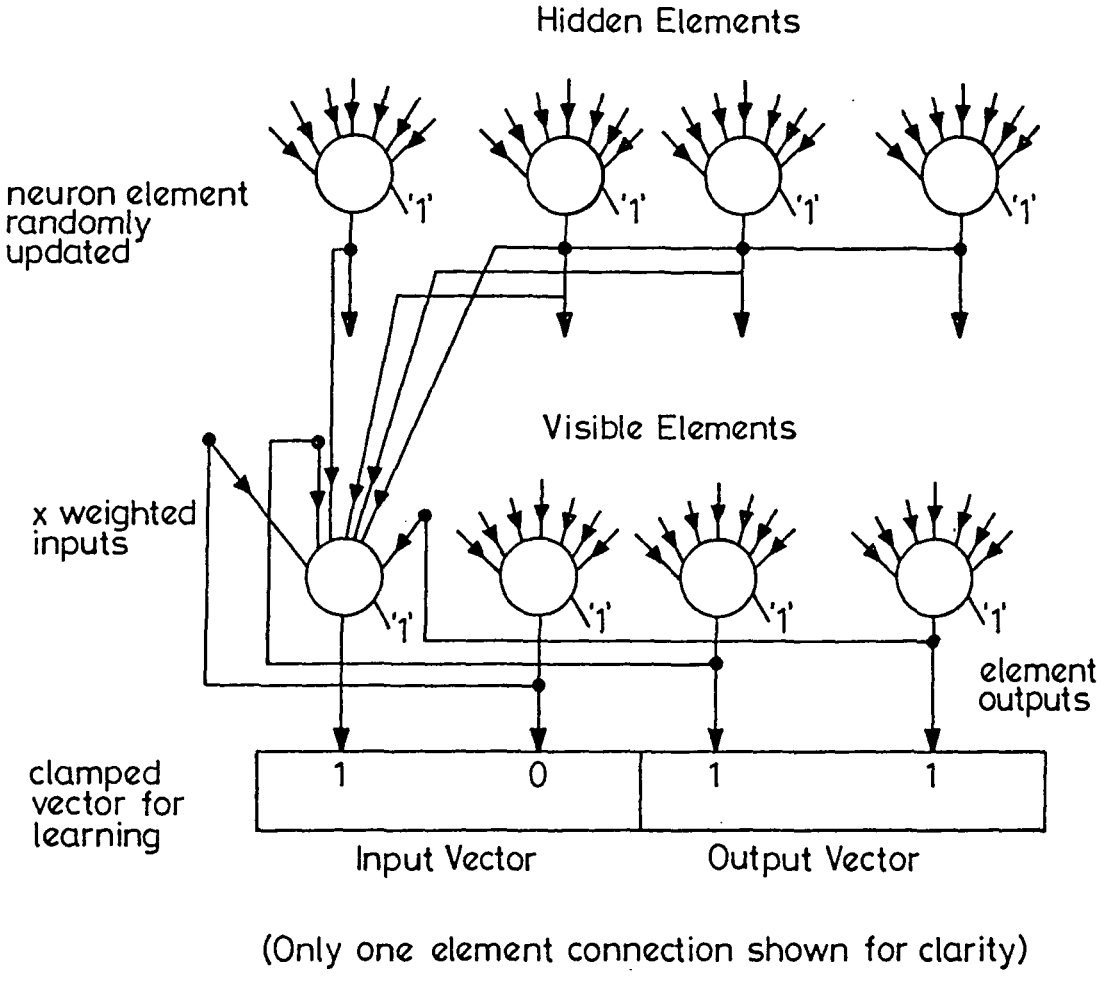


Figure 5.16 Boltzmann Machine

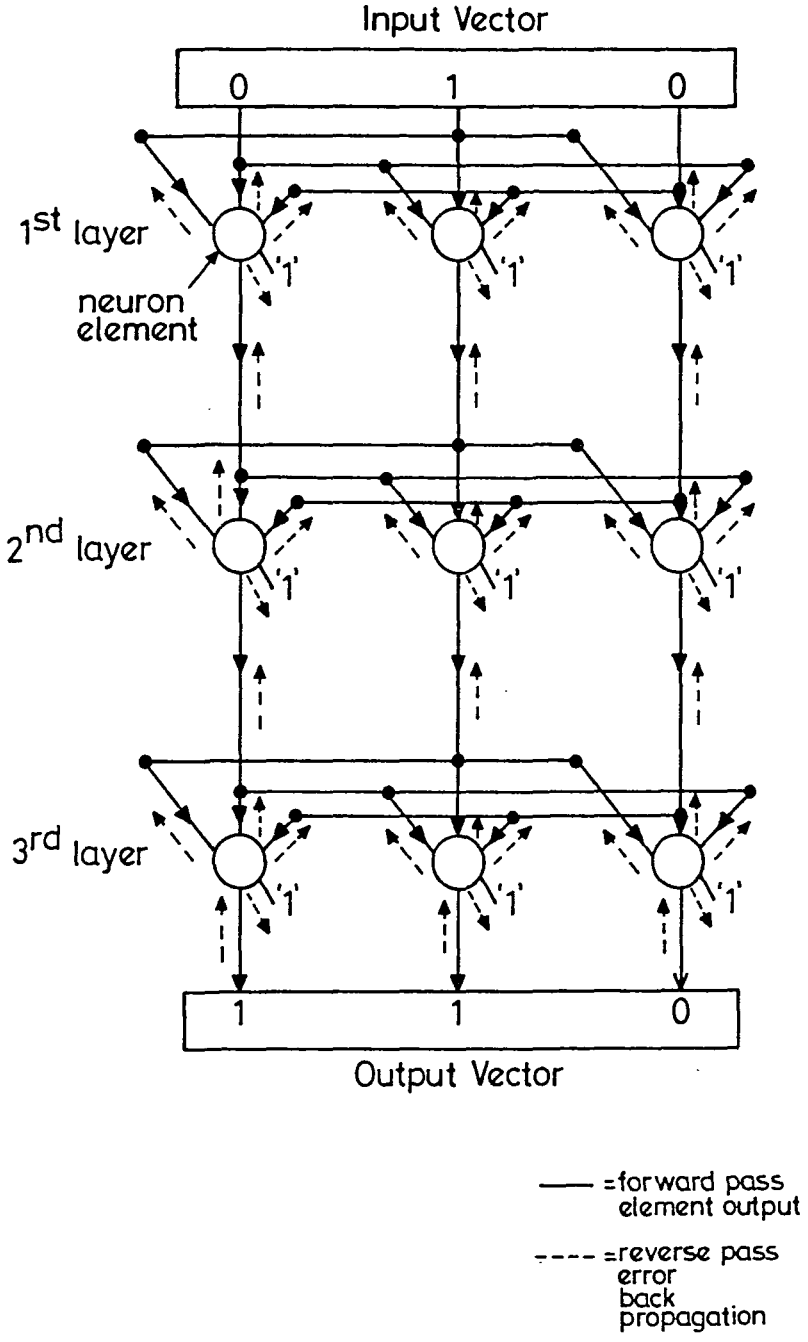


Figure 5.17 Error Backpropagation network

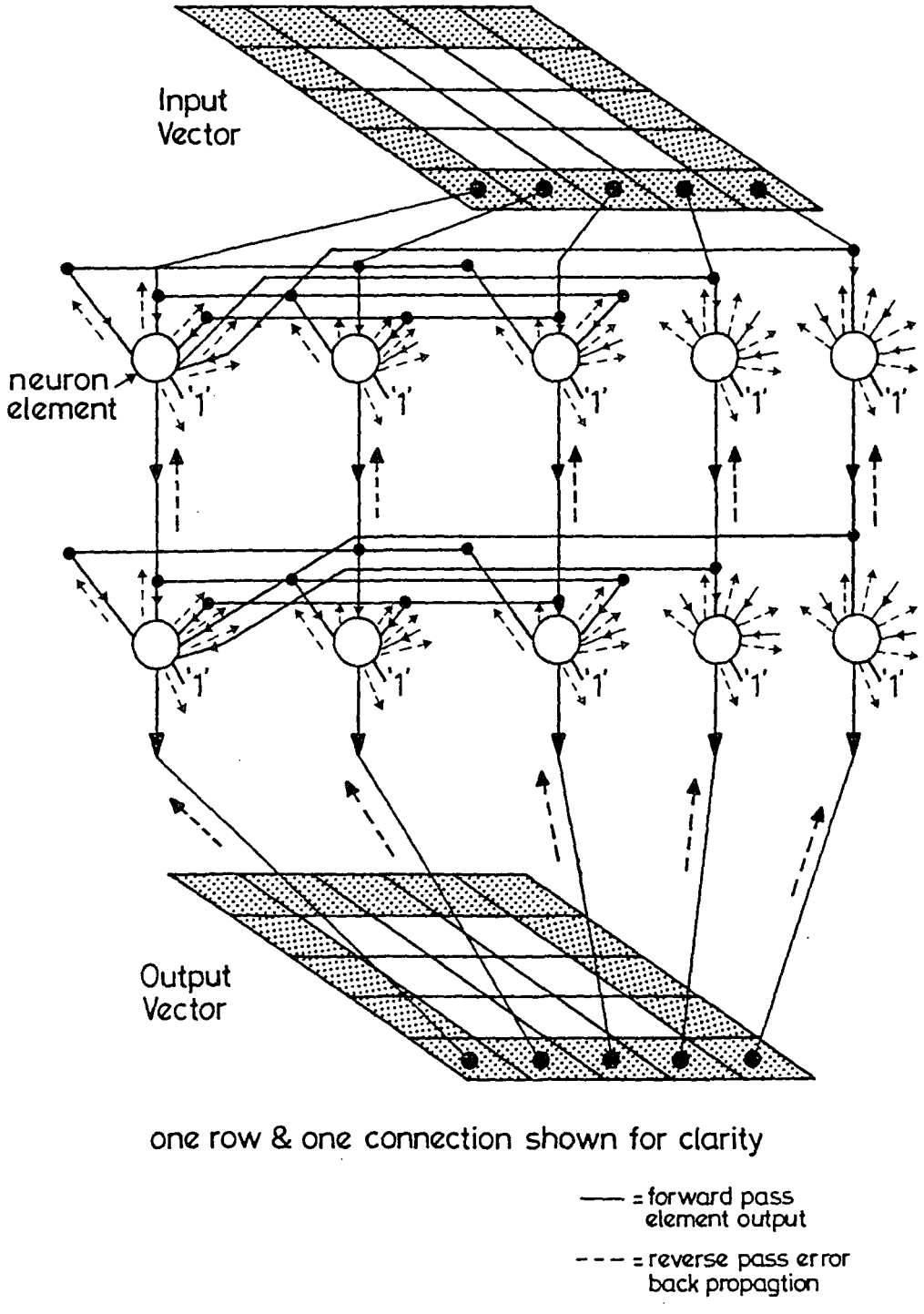


Figure 5.18 Error Backpropagation network – recognition of figure zero

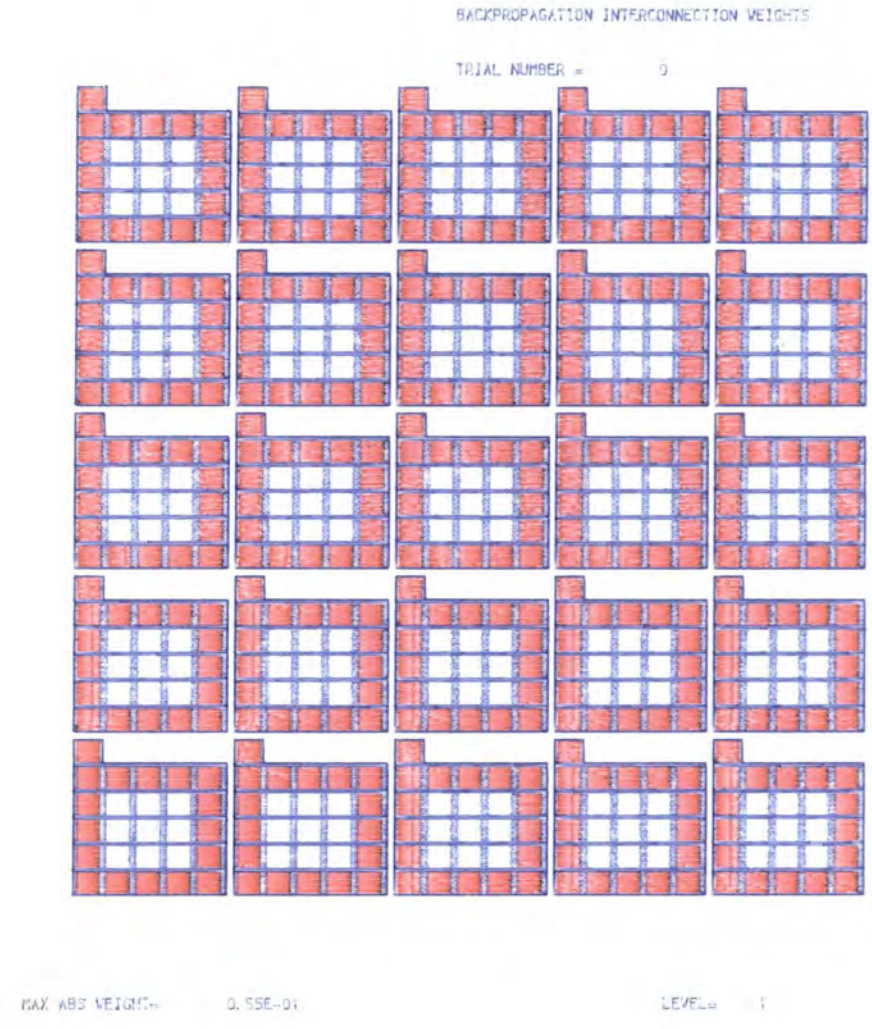


Figure 5.19a Level 1 Weight diagram 5 x 5 x 2 Backpropagation array

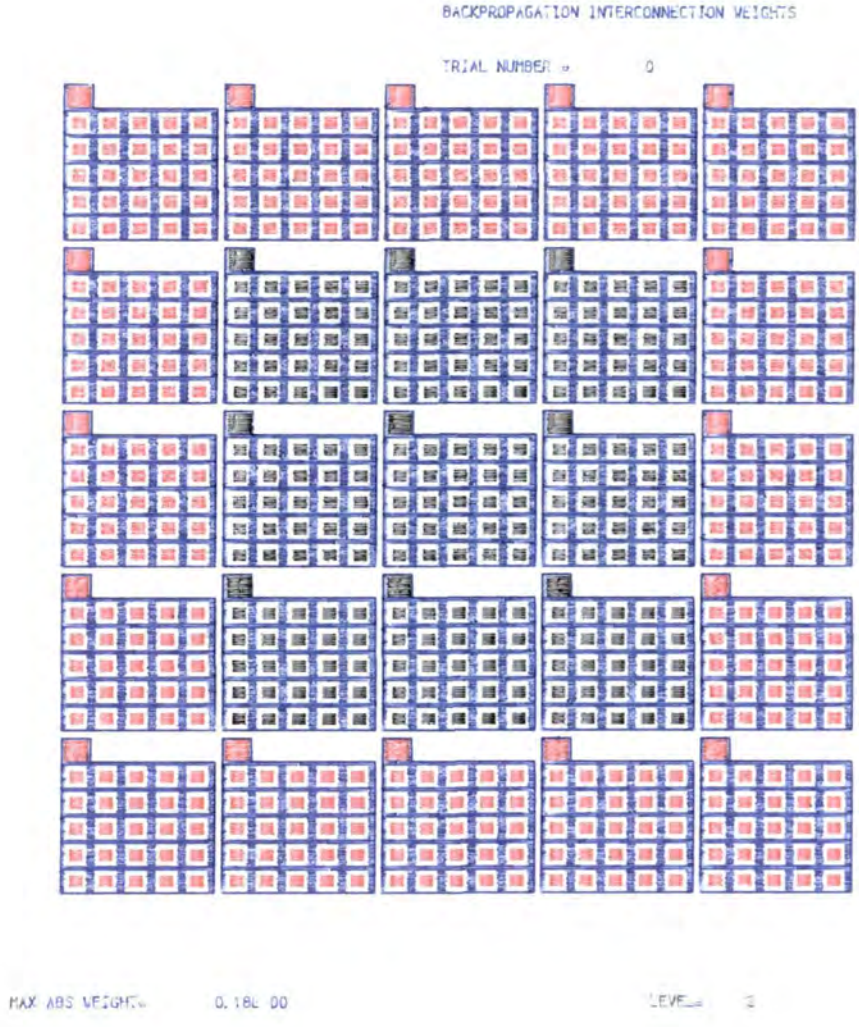


Figure 5.19b Level 2 Weight diagram 5 x 5 x 2 Backpropagation array

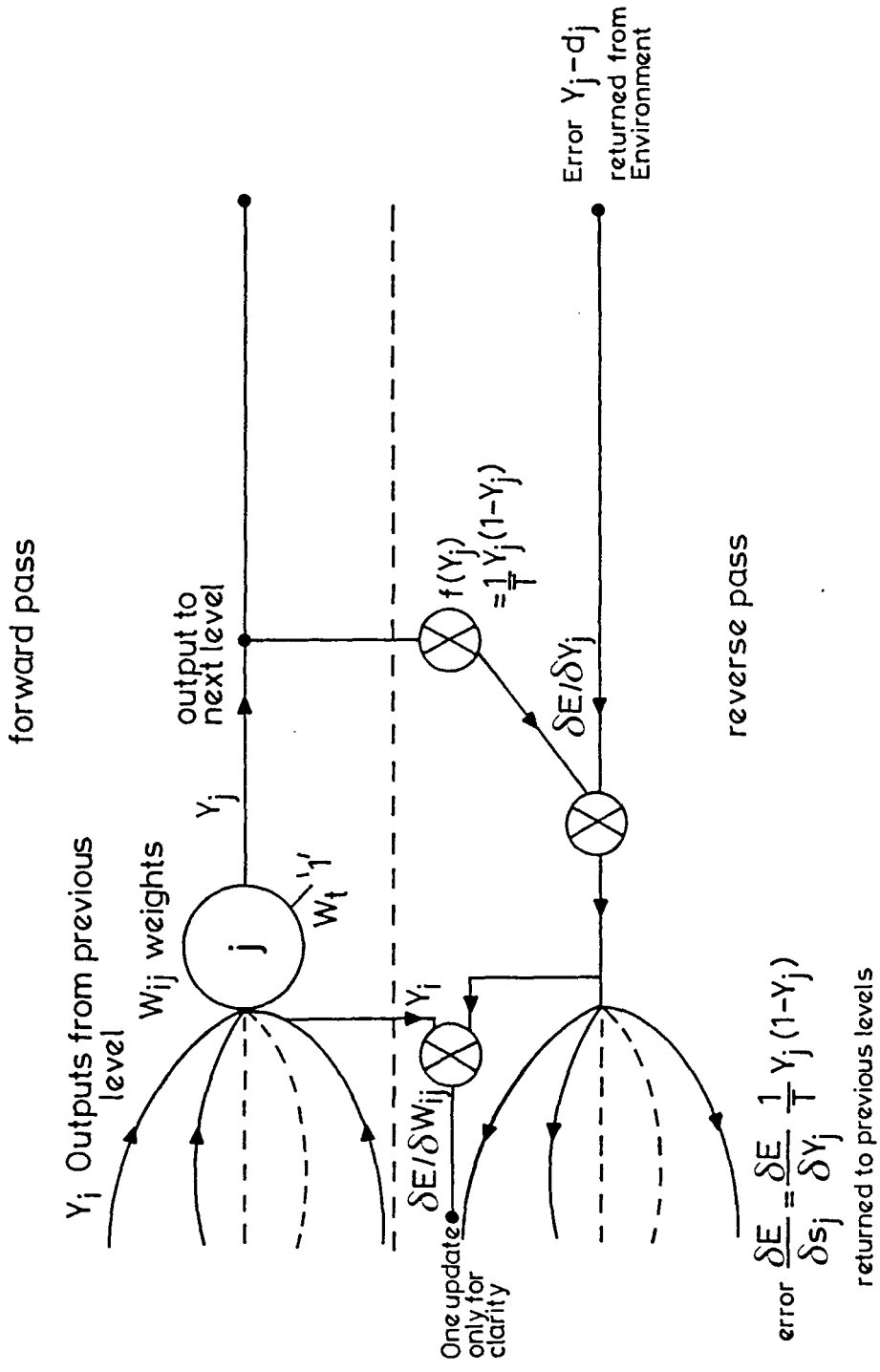


Figure 5.20a Error Backpropagation – Final Layer Element

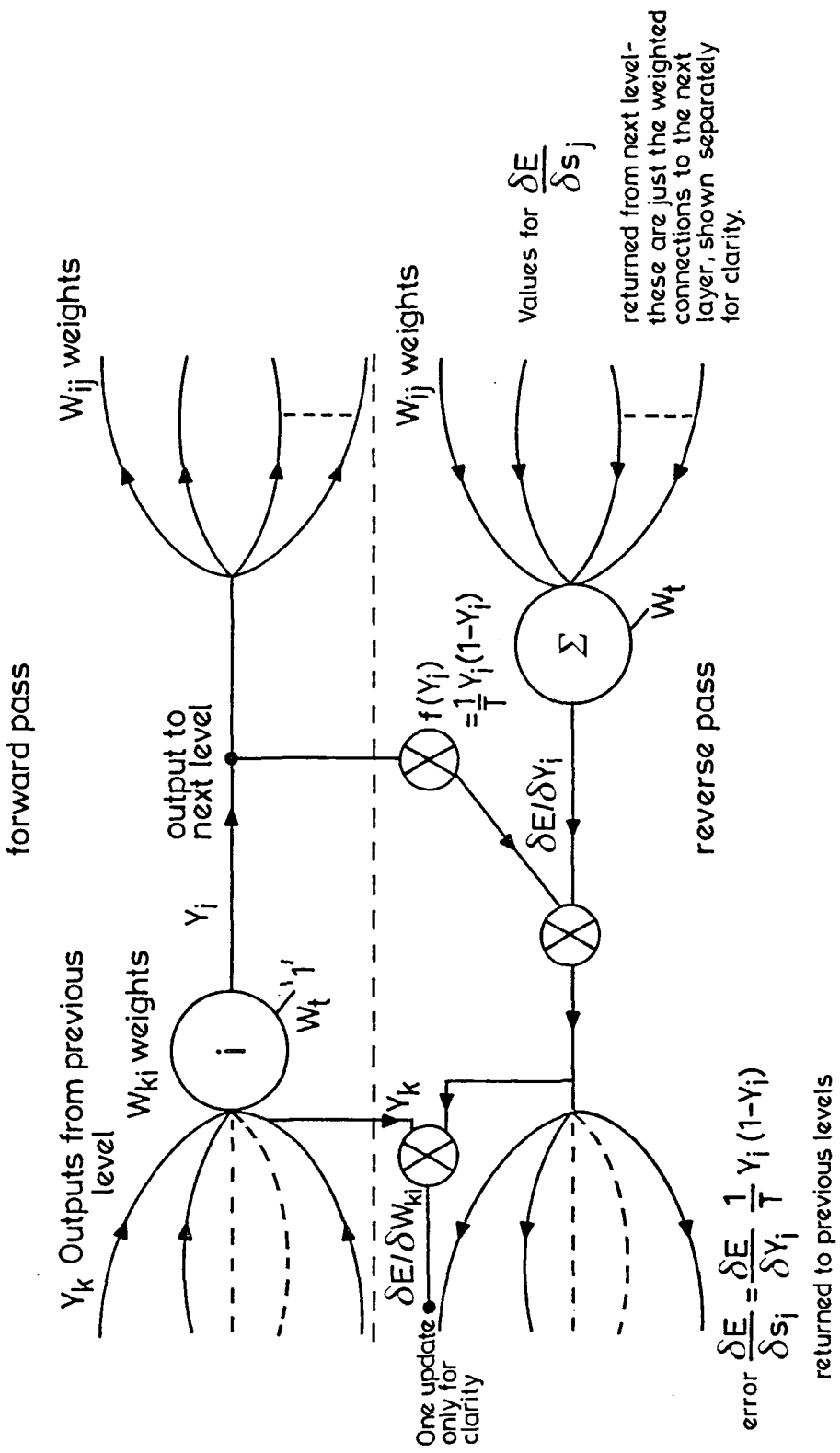


Figure 5.20b Error Backpropagation – Previous Layer Element

Table 5.1 - Logic element Truth Table

Input	Output
0,0	X
0,1	X
1,0	1
1,1	0

Table 5.2 - Exclusive OR Truth Table

Input	Output
0,0	0
0,1	1
1,0	1
1,1	0

Chapter Six

Associative Reward–Punish (A_{R-P}) Learning Automata

6.1 Introduction

The second half of this thesis draws together methods described in the first five chapters to form a novel series of stochastic networks based on learning automata using a reinforcement signal to adapt their response from an environment. Other similar approaches include the Widrow–Hoff 'Adaline' type adaptive element, [105], [106]. This chapter presents a learning automaton, known as the Associative Reward–Punish automaton or A_{R-P}, based on methods described in Chapter Four, which has a neural element–like structure as discussed in Chapter Five, [107], [108], [109]. This permits the element to perform pattern recognition together with the learning of connection weights using reinforcement or unsupervised learning, [110], [111], [112]. This method has the advantage that explicit error signals need not be presented. The entire network, including hidden elements may receive a global signal indicating success or failure and over a period of trials, converge to a correct solution for non–trivial problems. The limitations of this method of error feedback are considered and original alternatives proposed.

The chapter introduces the A_{R-P} element by initially considering the Linear Reward–Punish (L_{R-P}) algorithm introduced in Chapter Four and describing how the method may be adapted to enable pattern recognition. A

performance criterion is introduced and example simulations presented. It is shown how the L_{R-P} method may be extended to form the A_{R-P} algorithm.

Simulations for a single A_{R-P} element are performed using the previous parameters and the results compared. The success of the A_{R-P} algorithm leads to the study of multiple A_{R-P} elements working in cooperative teams. These are investigated with three main examples. The first example details a team of two A_{R-P} automata working together to solve a trivial pattern recognition problem. The second example shows two A_{R-P} automata in a different configuration solving the non-trivial Exclusive OR parity problem as discussed in Chapter Five and the final example shows a larger array of ten A_{R-P} automata solving the 4-2-4 Channel Bandwidth problem, [72]. This latter problem is a good example of the use of the hidden element layer. The programs used in these A_{R-P} simulations are documented in Appendix Five and a method for predicting single element A_{R-P} performance and weight convergence values in Appendix Six.

6.2 Pattern Recognition using Learning Automata

This section considers the adaptation of a learning automaton to permit pattern recognition. Consider the simple two-state automaton as described in Chapter Four. This automaton can take two actions such that it outputs a '1' or a '0'. Let p equal the probability of a '1' output and therefore the probability of a '0' is (1-p). The normal procedure for training an automaton would be to apply an Environment reinforcement as shown in Figure 6.1, and a learning algorithm such as L_{R-P}, such that the probability p

is adjusted to give the correct action from the automaton. This arrangement does not perform any kind of pattern recognition as there is no input to the automaton other than the reinforcement signal from the environment.

6.3 The L_{R-P} Element

Compare a simple neural element shown in Figure 6.2 with the automaton of Figure 6.1. For clarity, consider an element which has only two inputs plus a threshold input (as explained in Chapter Five) and required to respond to a set of inputs with outputs as shown in Table 6.1. There are then four summations possible, one for each of the input cases which involve the two input weights and the threshold weight.

It may be seen from Table 6.1 that for binary input vectors, there is a unique combination of weights for each case. Consider equations 6.1 and 6.2 which describe the weighted summation and subsequent conversion to a probability of firing, ie. output of a '1'.

$$s(t) = \sum_{n=1}^{N} x_n(t) w_n(t) \tag{6.1}$$

$$p(t) = \frac{1}{(1 + e^{-s(t)/T})} \tag{6.2}$$

It may be seen that the probabilities of firing p, for each input vector are effectively given by the summation of the appropriate combinations of weights where equation 6.2 ensures that the resulting probability never exceeds the limits 0 and 1, ie. saturates.

The problem reduces to four superposed automata, only one of which is 'enabled' for each input pattern although they share common weights.

Therefore, in order to adjust the probability of firing for each input case, it is merely necessary to adjust the weights concerned for each input pattern using an appropriate algorithm. Clearly the most simple updating rule would be of the L_{R-P} form where the probability of selecting a successful action is increased. Note that the probability measure is preserved by the limiting action of equation 6.2. This corresponds to updating the weights by an increment $\Delta w_n(t)$ given by;

$$\Delta w_n(t) = \begin{cases} \rho[y(t) - (1 - y(t))]x_n(t) & \text{if Success} \\ \rho\lambda[(1 - y(t)) - y(t)]x_n(t) & \text{if Failure} \end{cases} \tag{6.3}$$

This becomes simply,

$$\Delta w_n(t) = \begin{cases} \rho[2y(t) - 1]x_n(t) & \text{if Success} \\ \rho\lambda[1 - 2y(t)]x_n(t) & \text{if Failure} \end{cases} \tag{6.4}$$

ρ and λ are constants. λ defines the rate of learning. When $\lambda = 0$, this algorithm becomes a reward/inaction method.

All that equation 6.4 describes is that for a success or reward signal, the weights involved should be increased if the element output was a '1' and decreased otherwise. This is reversed for a failure or punish signal but with a smaller increment due to the λ constant. Note that one problem immediately apparent from this algorithm is that in some cases, weights may be continually increased or decreased, in other words, not converge since we rely on the saturation of the summation to preserve probability measure, rather than adjusting the weights to do this. The entire configuration, showing the element interacting with both the environment and the input pattern is shown in Figure 6.3. It is possible to evaluate the performance of this element as follows.

6.3.1 Performance Criterion

The performance of the element at time t, M_t, is defined as the expectation that reward will follow an action (as discussed in Chapter Four), and may be evaluated as follows. Consider the presentation of a single x input. Taken together with the threshold connection, there are effectively two input x vectors used, $x^{(1)} = (1,0)$ and $x^{(2)} = (1,1)$. Let the environment reward probabilities be as given by Table 6.2. Defining p_t^{0x} as the probability that the output of the element, y(t) = 0 and p_t^{+1x} as the probability that y(t) = +1.

$$\begin{aligned} M_t &= \sum_{x \in X} \xi^x [Pr\{r(t) = +1 \mid x(t) = x\}] \\ &= \sum_{x \in X} \xi^x [d(x,+1) p_t^{+1x} + d(x,0) p_t^{0x}] \end{aligned} \tag{6.5}$$

Denote p_t^{+11} as the probability that the element will emit action y(t) = +1 on trial t, for input vector $x^{(1)}$. Similarly, p_t^{+12} is the probability that y(t) = +1 for input vector $x^{(2)}$. It follows that:

$$\begin{aligned} p_t^{01} &= 1 - p_t^{+11} \\ p_t^{02} &= 1 - p_t^{+12} \end{aligned} \tag{6.6}$$

using equation 6.2;

$$p_t^{+11} = \frac{1}{1 + e^{-w_1/T}} \tag{6.7}$$

$$p_t^{+12} = \frac{1}{1 + e^{-(w_1+w_2)/T}} \tag{6.8}$$

Considering equation 6.5 and substituting values from Table 6.2:

$$\begin{aligned} M_t = \quad & 0.1 \;.\; p_t^{+11} + 0.9 \;.\; (1 - p_t^{+11}) \\ & + \; 0.9 \;.\; p_t^{+12} + 0.1 \;.\; (1 - p_t^{+12}) \end{aligned} \tag{6.9}$$

From equation 6.9, it is apparent that if the element chooses correctly to respond to $x^{(1)} = (1,0)$ with y(t) = 0 and to $x^{(2)} = (1,1)$ with y(t) = +1,

then the maximum performance achievable will be

$$M_{max} = \frac{0.9 + 0.9}{2} = 0.9 \tag{6.10}$$

6.3.2 L_{R-P} Simulations – One Element

The L_{R-P} element was simulated for the environment reward probabilities shown in Table 6.2. Figure 6.4a shows the performance M_t of a single L_{R-P} element against Number of Trials, for the arrangement of Figure 6.3. The simulation was performed for three different values of λ, with $\rho = 0.5$ and the system 'Temperature', T = 0.5.

$$\lambda_1 = 0.01$$
$$\lambda_2 = 0.05 \tag{6.11}$$
$$\lambda_3 = 0.25$$

M_t was averaged over 100 sequences of 10,000 trials. Figure 6.4b shows the values of the element weights, w_1, w_2 over the trials for the three values of λ. Figures 6.4c and 6.4d show M_t over 1 sequence of 10,000 trials.

Whilst the L_{R-P} simulation shown in Figure 6.4a shows the element successfully converging to the optimum value for $\lambda =$ 0.25, it is evident that for SMALLER values of λ, the algorithm has inferior performance, converging to values less than the optimum as expected.

Note that the weights fail to converge for $\lambda = 0.25$. This is because, as explained earlier, the weight updates are due only to the value of the reinforcement without taking into account the current weight values since the output probability function saturates as it approaches either end limit.

Consequently, for values of λ approaching ρ, it is possible for weights to continually increase. It will now be shown how the A_{R-P} algorithm successfully avoids this problem by reducing weight values appropriately where the output probability is approaching either limit.

6.4 The A_{R-P} Element

The A_{R-P} element is a learning automaton due to Barto, [107]. Barto's derivation proceeds as follows. The element is provided with an input x vector, $x_n(t)$ and performs the weighted summation as described by equation 6.1. The element chooses an action a(t) which takes on the value 1 with probability p given by equation 6.2 and -1 otherwise. This output is then zero-thresholded to produce a 1 or 0 output, y(t) to the Environment which rewards the element with probabilities as shown in Table 6.2.

Consider initially, two input x vectors, $x^{(1)} = (1, 0)$ and $x^{(2)} = (1, 1)$ from a set X. It will be noted that one of the inputs is always at '1' and this may be considered to be the threshold input in this example. These vectors are presented with equal probability at each trial t. In other words the probability of each vector occurring is 0.5 ($\xi^1 = \xi^2 = 0.5$). The two input A_{R-P} element forms the weighted sum at trial t, s(t) of the inputs as follows:

$$s(t) = \sum_{n=1}^{N} x_n(t).w_n(t) \tag{6.12}$$

where $w_n(t)$ are weights initially set to zero, and N=2. The weights are updated on each trial t. The element then chooses an action a(t) which is

+1 with probability p(t) and -1 otherwise, given by;

$$p(t) = \frac{1}{1 + e^{-s(t)/T}} \tag{6.13}$$

The output of the A_{R-P} element y(t) is zero-thresholded such that

$$y(t) = \begin{cases} +1, & \text{if } a(t) > 0 \\ 0, & a(t) < 0 \end{cases} \tag{6.14}$$

The Environment then examines the value of y(t) against the input x values and provides a reinforcement signal r(t) to the element. For each pattern x in X and each action y, the reinforcement signal is initially defined as:

$$r(t) = \begin{cases} +1, & \text{with probability } d(x,y) \\ -1, & \text{with probability } 1 - d(x,y) \end{cases} \tag{6.15}$$

The probabilities d(x, y) are as shown in Table 6.2.

It is now necessary to define the expectation $E\{a(t) \mid s(t)\}$ that the element will have action a(t) given weighted summation s(t). The probability that the element will have action $a(t) = +1$ is p_t^{+1x} ie.

$$p_t^{+1x} = \frac{1}{1 + e^{-s(t)/T}} \tag{6.16}$$

and

$$p_t^{-1x} = 1 - p_t^{+1x} \tag{6.17}$$

Defining the expectation $E\{a(t) \mid s(t)\}$ as

$$\begin{aligned} E\{a(t) \mid s(t)\} &= -1.p_t^{-1x} + 1.p_t^{+1x} \\ &= 2p_t^{+1x} - 1 \end{aligned} \tag{6.18}$$

$E\{a(t) \mid s(t)\}$ is plotted against s(t) in Figure 6.5 for three values of T, where T is the 'Temperature' of the system.

$$T_1 = 0.25$$

$$T_2 = 0.50 \tag{6.19}$$

$$T_3 = 1.00$$

Barto defines the A_{R-P} element weights $w_n(t)$ updating algorithm as follows, [107].

$$\Delta w_n(t) = \begin{cases} \rho[r(t)a(t) - E\{a(t) \mid s(t)\}]x_n(t) & \text{if } r(t) = +1 \quad \text{(Success)} \\ \lambda\rho[r(t)a(t) - E\{a(t) \mid s(t)\}]x_n(t) & \text{if } r(t) = -1 \quad \text{(Failure)} \end{cases} \tag{6.20}$$

ρ and λ are constants. λ defines the rate of learning. When $\lambda = 0$, the A_{R-P} algorithm becomes a reward/inaction method. By noting that the reinforcement signal r(t) is defined for this algorithm as +1 for a reward signal and −1 for punishment then rewriting equation 6.20 in terms of the output y(t) gives;

$$\Delta w_n(t) = \begin{cases} \rho[2y(t) - 1 - E\{a(t) \mid s(t)\}]x_n(t) & \text{if Success} \\ \rho\lambda[1 - 2y(t) - E\{a(t) \mid s(t)\}]x_n(t) & \text{if Failure} \end{cases} \tag{6.21}$$

Equation 6.21 for the A_{R-P} should be compared against equation 6.4 which describes the weight update for the L_{R-P} element. The only difference is that the weight update is reduced by the expectation of output of the element in the A_{R-P} case. This reduces the values of updates where the weights are such that the output probability is approaching saturation and thereby provides a bound.

6.4.1 A_{R-P} Simulations – One Element

An example of single A_{R-P} element operation is shown in figure 6.6a. This figure shows the performance M_t of the element against Number of Trials, for the arrangement of Figure 6.3 with reward probabilities as shown in Table 6.2. The simulation was performed for three different values of λ, with $\rho = 0.5$ and the system 'Temperature', T = 0.5.

$$\begin{aligned} \lambda_1 &= 0.01 \\ \lambda_2 &= 0.05 \\ \lambda_3 &= 0.25 \end{aligned} \tag{6.22}$$

M_t was averaged over 100 sequences of 10,000 trials. Figure 6.6b shows the values of the element weights, w_1, w_2 over the trials for the three values of λ. Figures 6.6c and 6.6d show M_t over 1 sequence of 10,000 trials. A method of predicting the convergence values for a single A_{R-P} element is described in Appendix Six and the dotted lines on the figures represent these values. By comparing these simulations with the L_{R-P} figures, it is apparent that the weights successfully converge to the predicted values. The success of the A_{R-P} method prompts the study of teams of such elements to investigate cooperative behaviour.

6.4.2 A_{R-P} Simulations – Two Elements

A suitable arrangement of two A_{R-P} elements is shown in Figure 6.7. A_{R-P} Element 1 is presented as before with input x vectors as training patterns but has no direct connection to the Environment. As before, one input of the training set always remains at '1' and this may be considered as

the threshold input. A_{R-P} Element 2 receives the output of the first element (either 1 or 0) but does not have direct connection to the input vector and has a threshold input. Element 2 makes a decision based on Element 1's output and then performs its action to the Environment. The Environment responds with a reinforcement signal to both elements simultaneously using the reward probabilities of Table 6.2.

Clearly there is no direct path from input pattern to the Environment and so the two elements must learn to cooperate in order to maximise their reward. The problem itself remains a linear separation problem as described in Chapter Five, for which one element would perform equally well. The results of this two element linear problem, taken over one sequence of trials are shown in Figures 6.8(a-c). Note that the weights are labelled for each element, e.g. w_{13} is w_3 of Element 1 and similarly w_{21} is w_1 of Element 2. This example is non-taxing in that it is a simple linear classification problem, using cooperating A_{R-P} elements where one would have been sufficient.

6.4.3 A_{R-P} Simulations – Exclusive OR Problem

The third example problem, again addressed to a 2 A_{R-P} element configuration, was that of learning to perform an Exclusive OR operation. This is the simplest case of the even parity check and requires 4 input training vectors, $x^{(0)} = (0,0)$, $x^{(1)} = (0,1)$, $x^{(2)} = (1,0)$ and $x^{(3)} = (1,1)$. The reward probabilities are shown in Table 6.3 and the 2 element A_{R-P} configuration used, in Figure 6.9. In this configuration, both elements are presented with the input vectors, but element 2 additionally has the output of element 1 as

an input. In addition to these inputs, both elements have a threshold input. Therefore element 1 is adjusting three weights whilst Element 2 is adjusting four weights.

As explained in Chapter Five, the Exclusive OR problem is a non-trivial problem requiring more than one element. By using two A_{R-P} elements which learn to cooperate, a maximum performance of 0.9 can be obtained. The results of the simulation are shown in Figures 6.10(a-e) with parameters as labelled taken over one sequence using a single value for λ, $\lambda = 0.01$ for clarity.

6.4.4 A_{R-P} Simulations – 4–2–4 Encoder Problem

In this final example, the use of an A_{R-P} network to solve the Channel Encoder problem is shown, [72]. The array consists of three levels of A_{R-P} automata. The first and third level have N elements and the second layer has $\log_2 N$ elements as shown in Figure 6.11. The object is to reproduce an N bit vector presented to the input layer at the output layer. The vector has the limitation that only one bit may be in the ON state and consequently there are only N combinations possible.

Since the second layer only has $\log_2 N$ elements, the array must develop the optimum coding scheme to allow information to pass through the second layer and be reproduced correctly. Where N=4, this is known as the 4–2–4 encoder, and for N=8, this is known as the 8–3–8 encoder. The problem becomes progressively more difficult for increasing N. Figure 6.12 shows the weights for the 4–2–4 encoder successfully solved by an A_{R-P} array

using reward probability=0.9, punish probability=0.1, $\rho = 0.5$, $\lambda = 0.01$, and T = 0.5.

6.5 Reinforcement Methods

Initial experiments involving the use of large arrays of A_{R-P} elements revealed that the method by which reinforcement was supplied was a critical factor. There have been some theoretical examinations of the reinforcement process in the literature, [110], [111], [112]. Practically, three major methods of encoding the probabilistic reinforcement signal were identified. These were: Success/Failure, Reciprocal and Gradient.

Success/Failure, was the method initially used in the A_{R-P} simulations. The array was rewarded with a probability p if correct and lesser probability q if incorrect, where q is not necessarily 1 - p. Reciprocal reinforcement, was defined as encoding the error cost function shown in equation 6.23 as inversely proportional to the probability of reward.

$$E = \frac{1}{2} \sum_{c} \sum_{j} (y_{j,c} - d_{j,c})^2 \tag{6.23}$$

In this case, maximum reward is achieved for minimum error, corresponding to a solution. The third method, Gradient reinforcement, involved use of the differential of the same error cost function such that the array was rewarded with probability=0.9 if the current error for a particular input was less than the previous error for the same input vector and rewarded with probability=0.1 otherwise.

When adopting this latter method for use in training A_{R-P} networks, 'lock-on' at a solution was ensured by rewarding with probability 1 when the correct response was achieved. However, the following team simulations were performed without this addition in order to gauge its necessity. Note the distinction that whilst the error backpropagation algorithm requires a **local** measure of the error at each of the output elements, the A_{R-P} array receives **global** reinforcement containing overall error gradient information.

6.5.1 Team simulation with different reinforcement schemes

The relative merits of each of the three reinforcement methods were investigated using a four element, two layer (ie. eight elements) A_{R-P} automata team as shown in Figure 6.13. The object of the team is to move from the bottom left of the 'board' shown in Figure 6.14, to the closest position to a stationary target positioned at the top right with no prior knowledge of the task. The criterion for reinforcement was based on the straight line distance between the array position and the target.

The starting position of the A_{R-P} array and the position of the target were each marked with an asterisk on the figures. The 'board' itself comprised 100 x 100 possible positions and the array was allowed only 10,000 trials. The array elements each received only a '1' constant input irrespective of position, in addition to reinforcement, and each of the four outputs corresponded to one square movement in the North, South, East or West directions. eg. if the output was 1000, then the array position moved 1 square North, 1100 resulted in no net movement, and diagonal movement was obtained from other

combinations.

Reinforcement was probabilistically supplied from the environment according to one of the three methods described earlier. The array had to learn to relate action to reinforcement to arrive at the correct solution, represented by the coincidence of array and target positions. Note, it was not sufficient for the array to locate itself at the target position, it had to also maintain it by 'locking–on' if this approach was to be used for the learning of input–output associations.

Figure 6.15a shows the movement of the array using the Success/Failure method with reward probability=0.9 for success and reward probability=0.1 for failure. Figure 6.15b is a plot showing the actual distance against number of trials. From the figure, it is apparent that there is considerable random movement about the 'board' before the array position coincides with that of the target and success reinforcement occurs with probability 0.9. In this case, the array has locked on to this successful move and only slight motion away from this position occurs. This was not found to occur with every simulation run however. Consider the potential number of moves available, with only one correct position (the target location) out of 10,000 possible positions. In the unlikely event that the array finds the correct location, the probability of reward is 0.9 which means that even when the array is successful, there is still a 0.1 probability of punishment which could prevent lock–on. For larger problems, it appears unlikely that the Success/Failure reinforcement scheme would be efficient in terms of learning speed since the chance of locating the correct response would become relatively small.

By comparison Figure 6.16a shows the movement of the array using the Reciprocal method with the reward probability being equal to 1 at minimum separation, and 0 at maximum separation. Figure 6.16b, as before is a plot showing the actual distance against number of trials. This seems a reasonable approach since there is more information in the reinforcement signal. However, the array failed to locate the target. The problem lay in the difficulty in observing degrees of success from the small changes in reinforcement probability.

Figure 6.17a shows the movement of the array using the Gradient method with a reward response with probability 0.9 from the environment for a move resulting in decreasing distance error and a reward probability of 0.1 otherwise. Figure 6.17b shows the actual distance against number of trials. Clearly the array learned extremely quickly. Note how, at the solution point, the array moved around the correct position but always returned. This was since there was still the probability of a punish signal even when the array was successful. This could be avoided by rewarding with probability one when the array is at the correct solution. It will be described later how the gradient method has been successful in training large multilayer A_{R-P} networks to compute non-trivial functions. The one minor drawback is that the error for each input–output case must be stored by the Environment to enable subsequent reinforcement to be made.

6.5.2 Future Work – Virtual Simulated Annealing

In the previous section, it was discussed how a gradient following reinforcement scheme could be successfully applied to large scale learning problems involving teams of automata. It was further described how, if the collective response of the array was to reduce the error, then it was rewarded with probability 0.9 otherwise with probability 0.1. In the event of zero error, the array was rewarded with probability one. This leads to a potentially interesting case.

Consider at trial 1, commencing all environment rewards equally at a probability of 0.5. Allowing the array to learn using the gradient method now has no clear direction since both increasing and decreasing error are rewarded equally. Now consider the situation where the reward probability is increased from 0.5 to 1.0 in small increments. In this way, the array will begin by being allowed to make moves which may equally well permit increasing and decreasing error but as the trials increase, mainly moves decreasing error would occur. The gradient method described in the last section is clearly a special case of this. In this way, the adjustment of the environment reward probabilities may be seen to be introducing a type of simulated annealing process as described in Chapter Five, [93]. The major distinction and advantage is that the gradient of the probability function of the element output is not altered, as in the Boltzmann machine and similar arrangements. Instead it is the single, global environment response which is altered. This may have considerable hardware and control advantages.

6.6 Conclusions and Summary – Chapter Six

This chapter has shown how learning automata may be developed into neural networks in order to perform pattern recognition and process information. The main advantage of this approach over conventional hierarchies of automata is that in the neural network no processors are idle, all take part in the collective decision. A particular type of pattern recognising automata, due to Barto et al., [107], has been studied and simulation results presented. Novel methods of training large groups of such automata have been presented which involve special but trivial coding of the reinforcement signal to carry more meaningful information.

Future work has been suggested which introduces an original analogy with simulated annealing, removing the necessity for alteration of the elements' response characteristics during the annealing schedule. Instead it is the global environment signal which is altered. This is likely to be much more convenient in a practical implementation. It would be interesting to examine the possible use of similar approaches to other error gradient techniques such as error backpropagation. The next chapter considers novel stochastic implementations of neural networks such as A_{R-P} learning arrays, using methods introduced in Chapter Two.

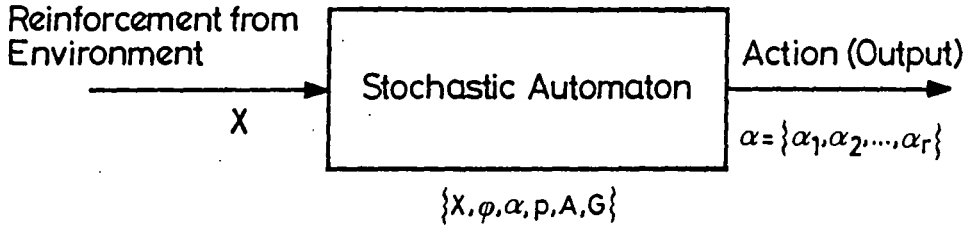


Figure 6.1 Stochastic Learning Automaton

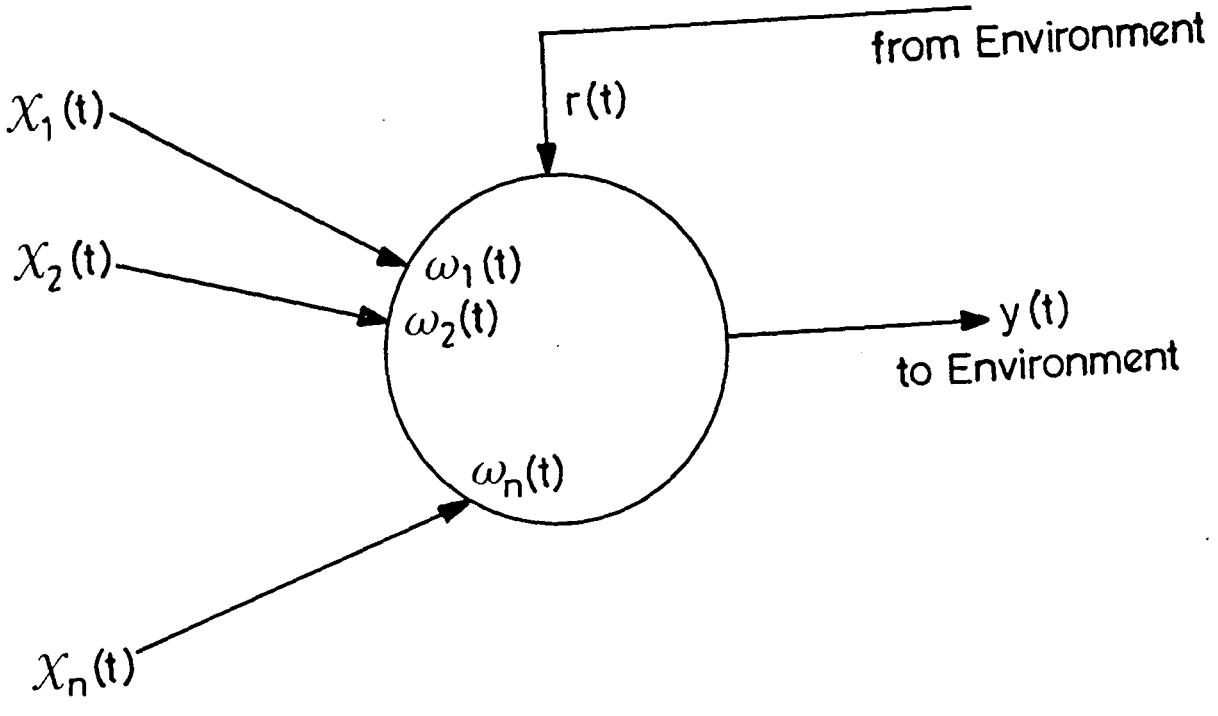


Figure 6.2 Pattern Recognising Automaton

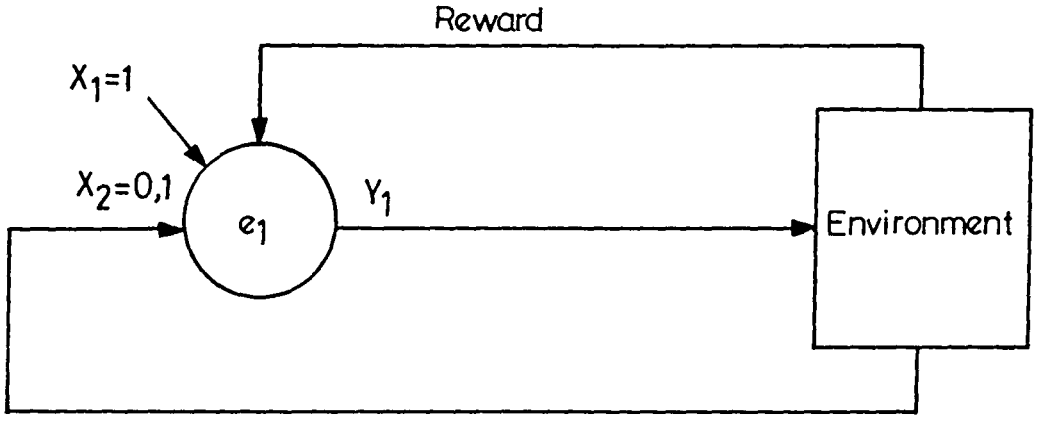


Figure 6.3 Automaton, Input and Environment

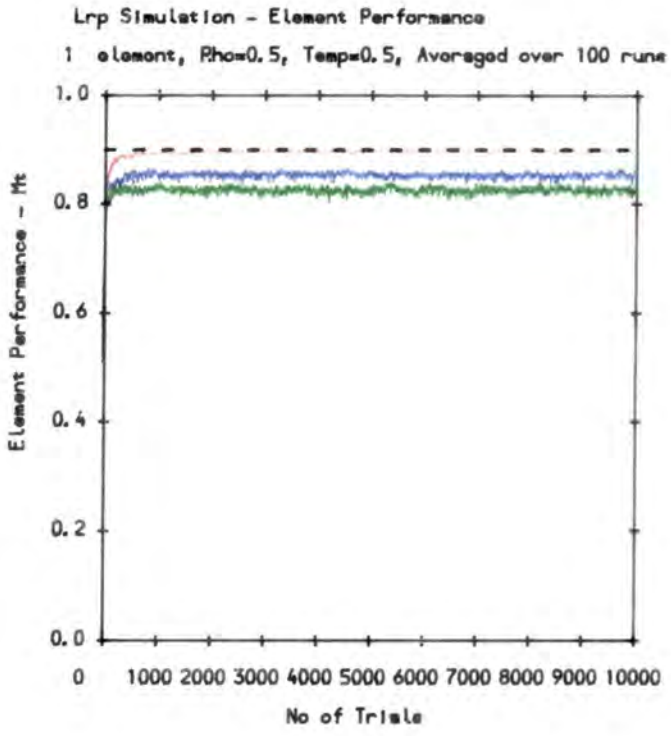


Figure 6.4a - M_t vs Number of Trials

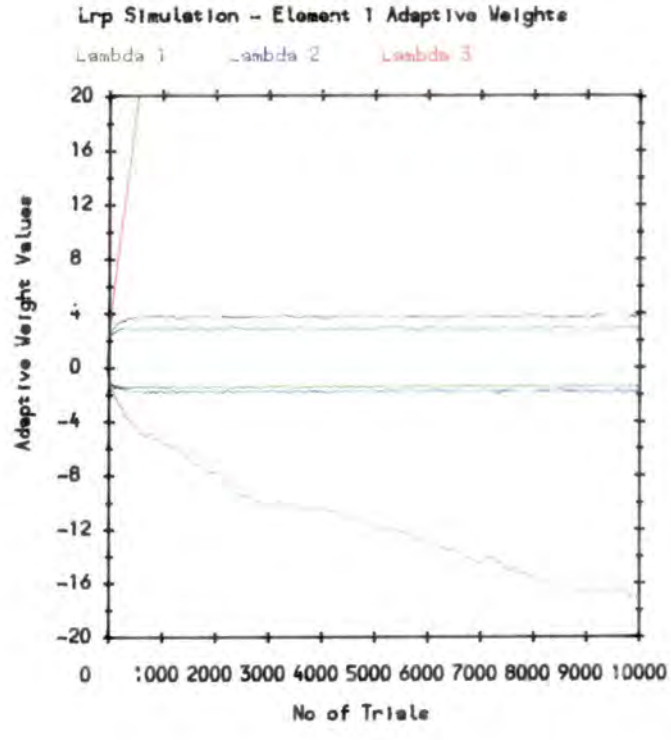


Figure 6.4b - w_1, w_2 vs Number of Trials

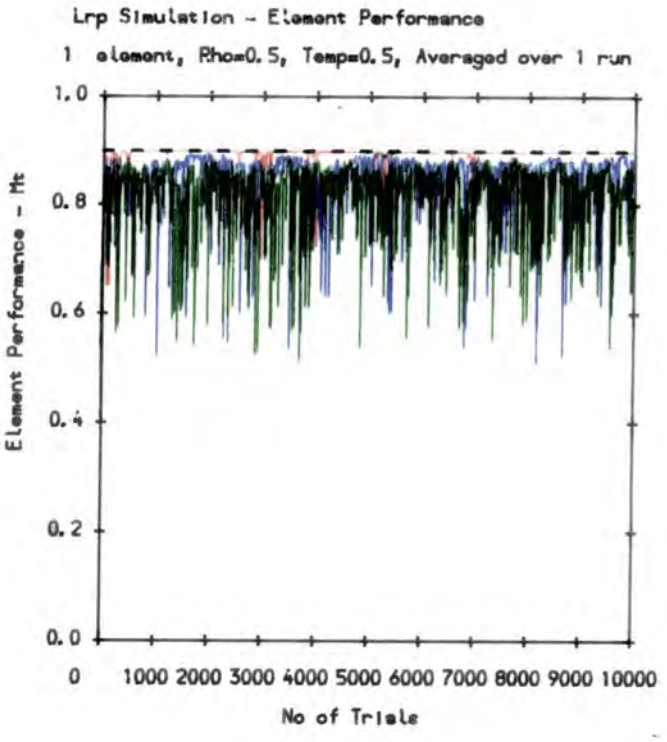


Figure 6.4c - M_t vs Number of Trials

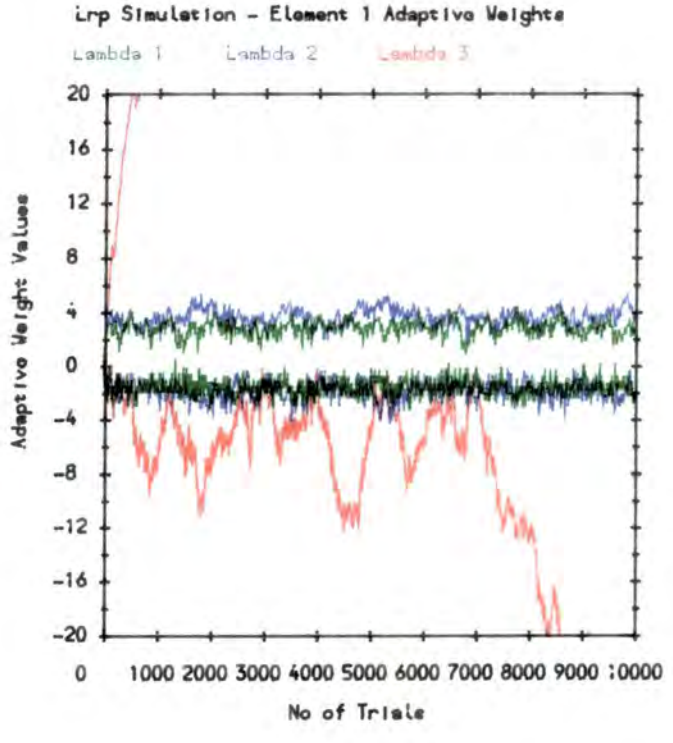


Figure 6.4d - w_1, w_2 vs Number of Trials

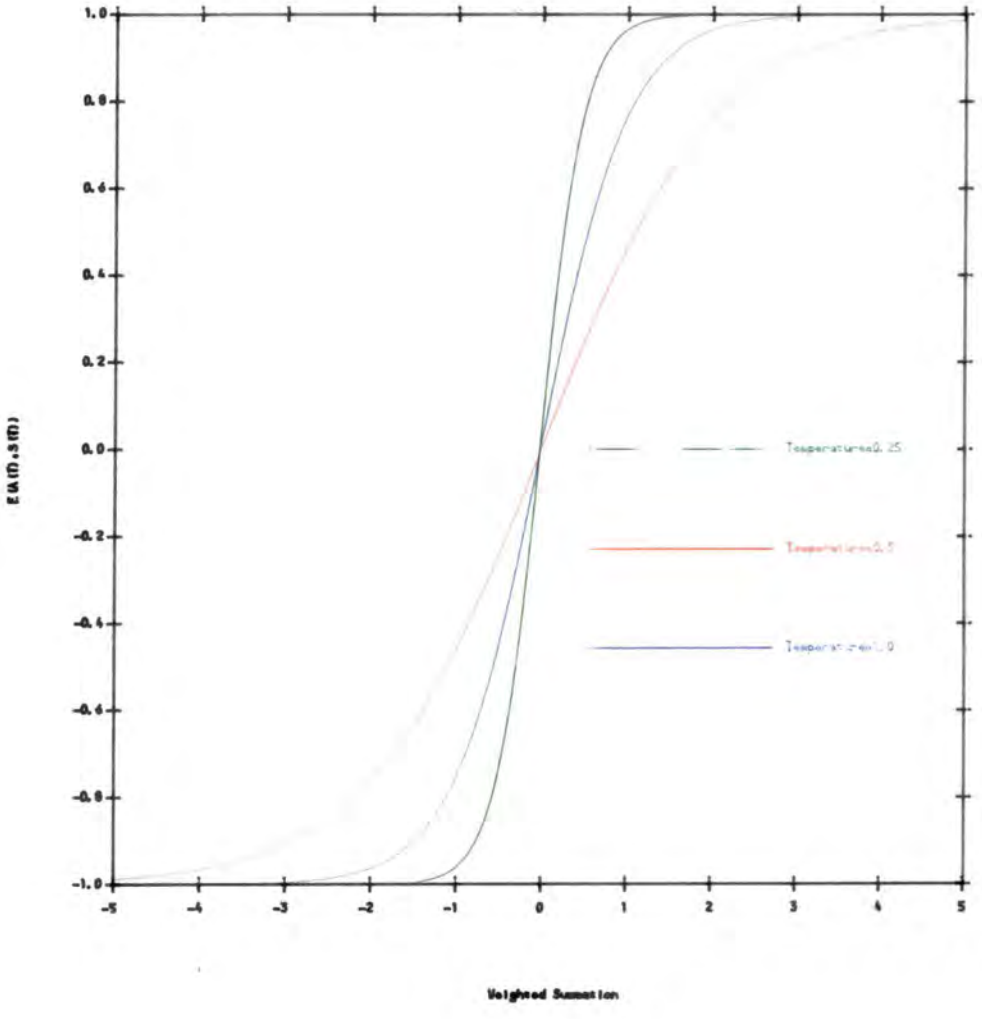


Figure 6.5 Expectation $E\{a(t) \mid s(t)\}$

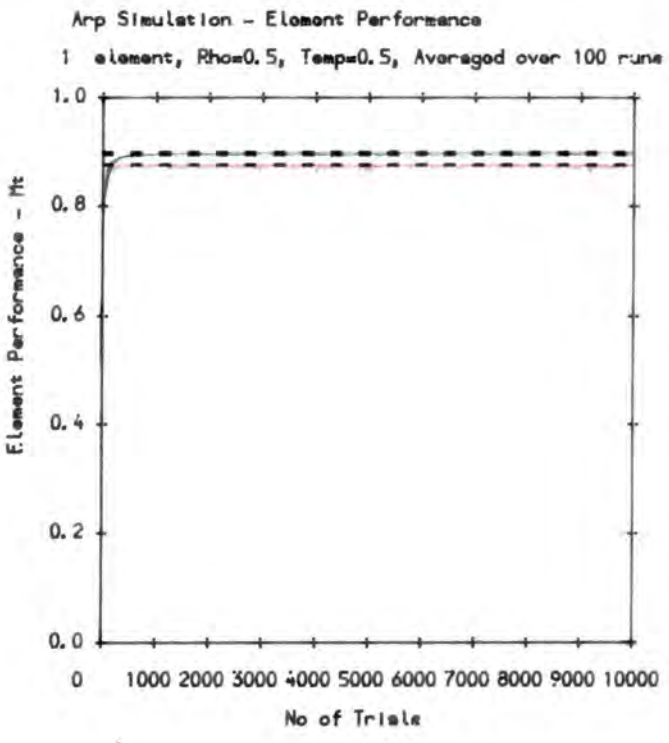


Figure 6.6a - M_t vs Number of Trials

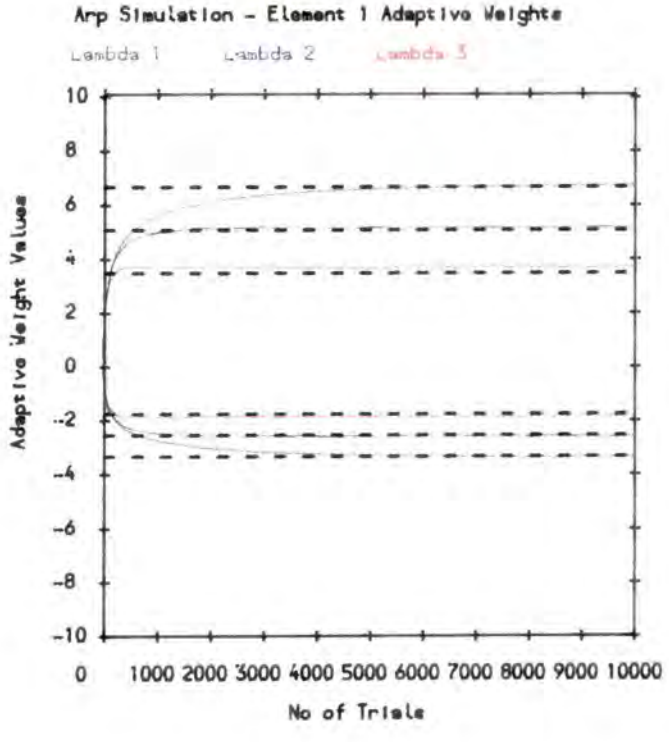


Figure 6.6b - w_1, w_2 vs Number of Trials

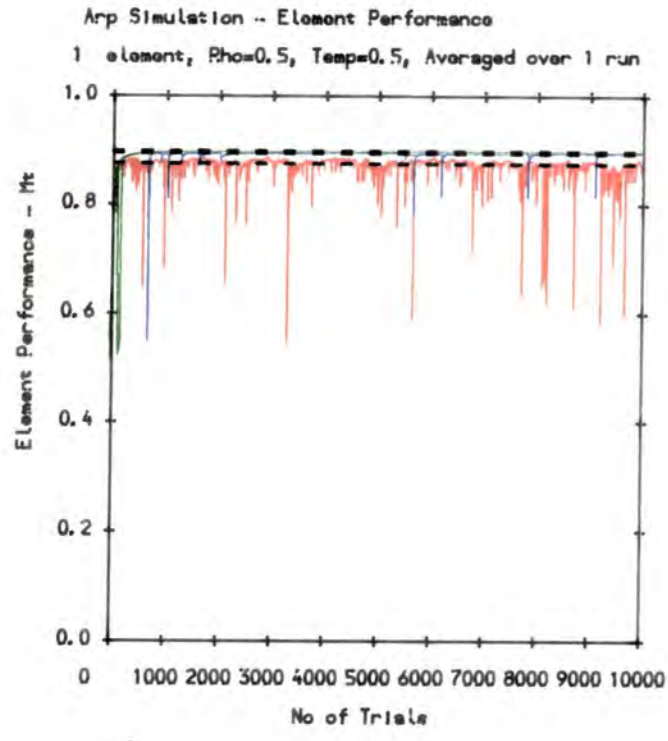


Figure 6.6c - M_t vs Number of Trials

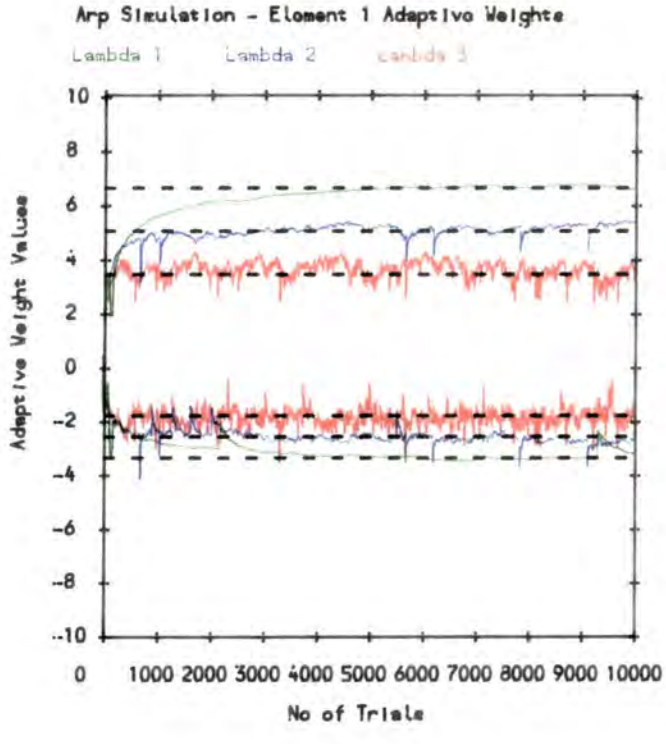


Figure 6.6d - w_1, w_2 vs Number of Trials

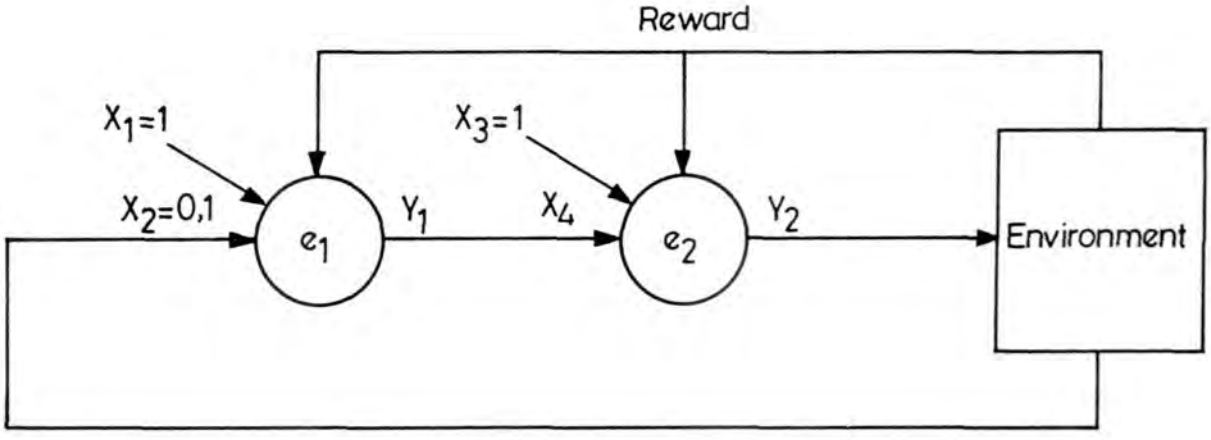


Figure 6.7 - 2 A_{R-P} Elements - Linear configuration

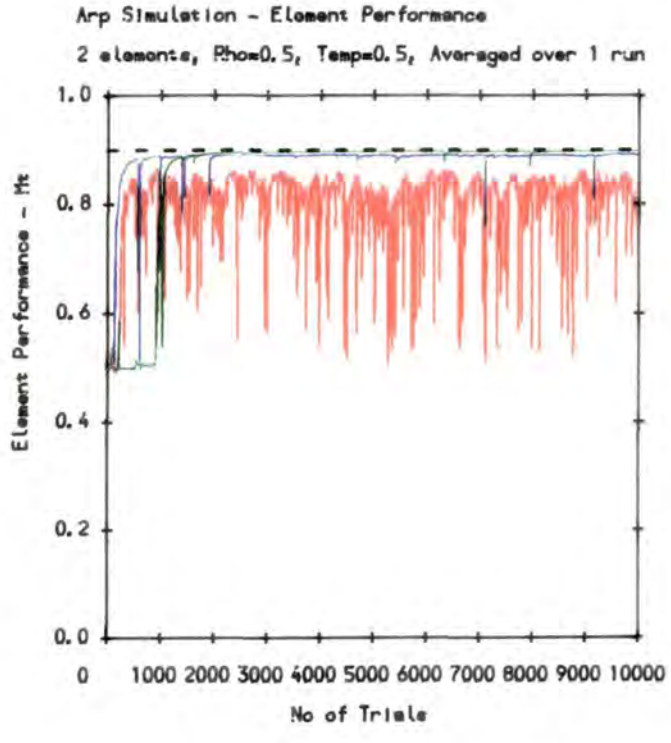


Figure 6.8a - M_t vs Number of Trials

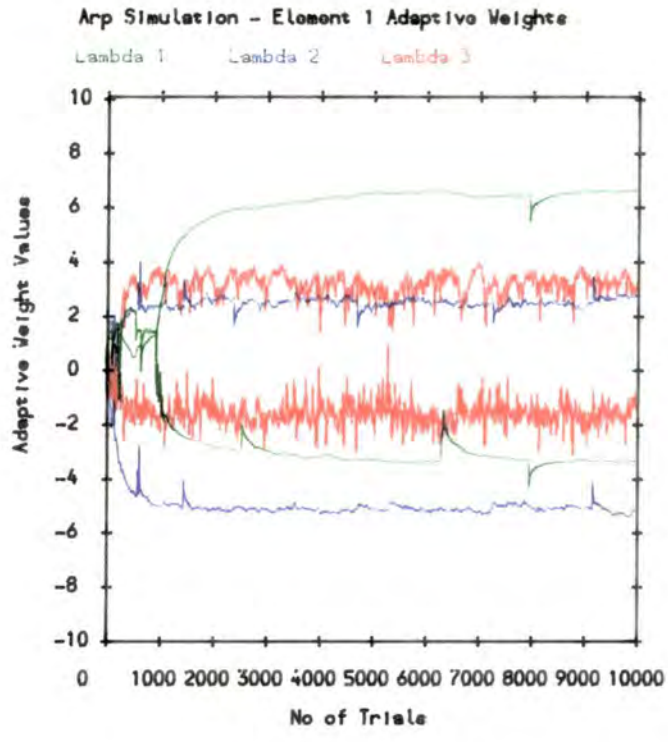


Figure 6.8b - w_{11}, w_{12} vs Number of Trials

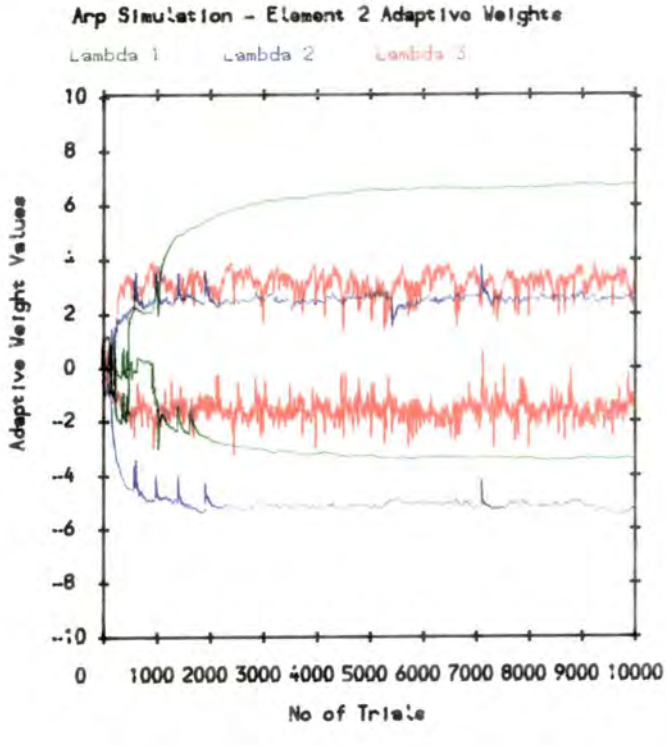


Figure 6.8c - w_{21}, w_{22} vs Number of Trials

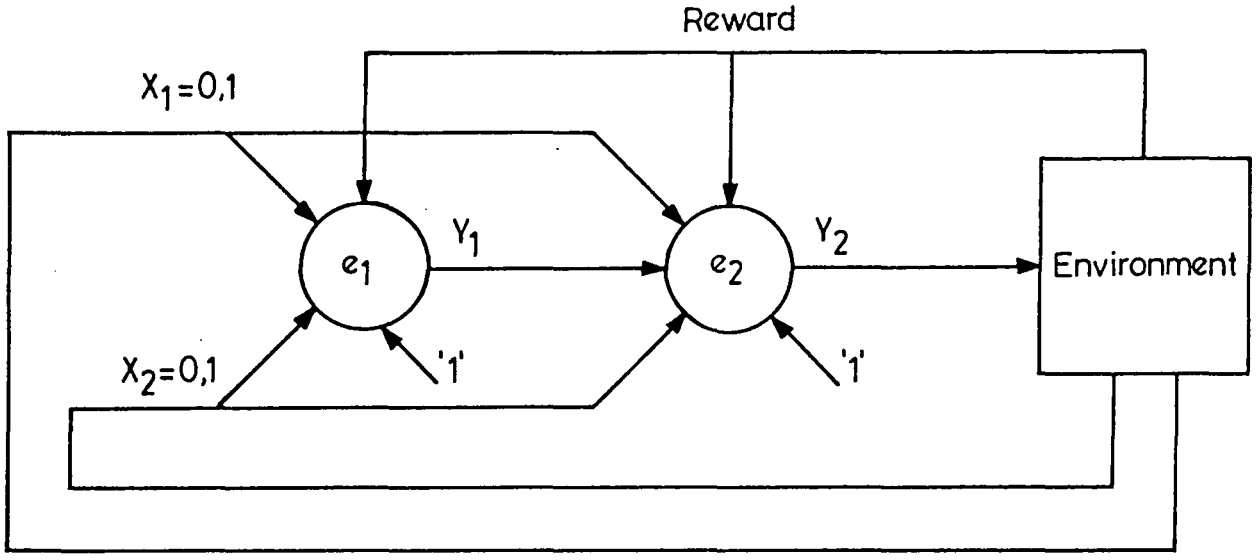


Figure 6.9 - 2 A_{R-P} Elements - parallel configuration

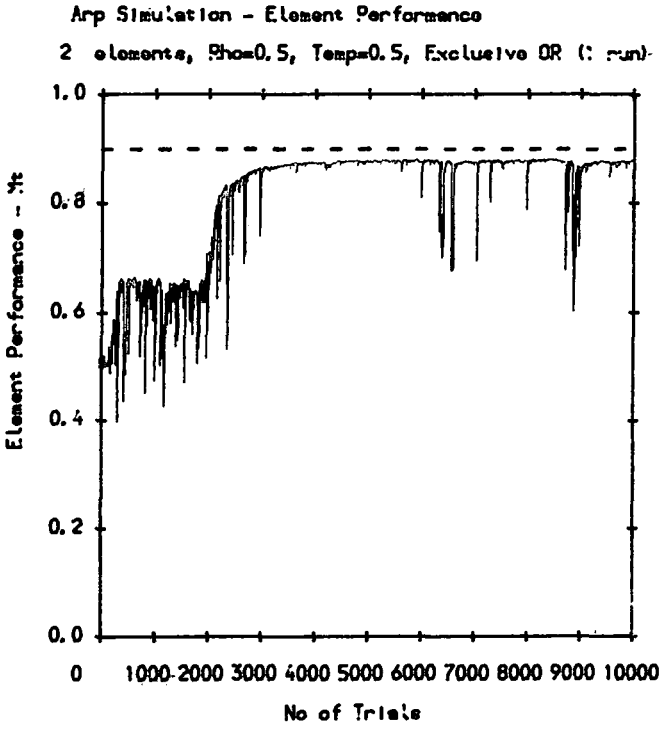


Figure 6.10a - M_t vs Number of Trials

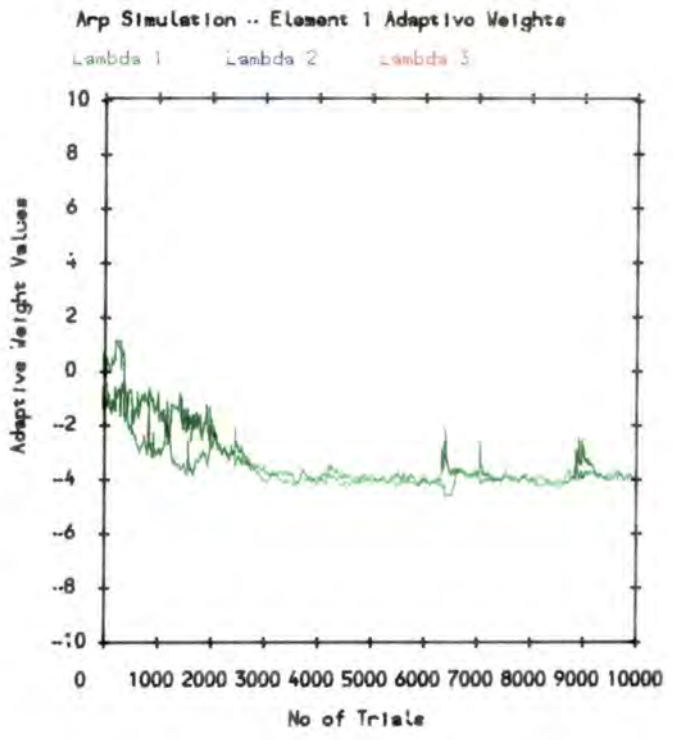


Figure 6.10b - w_{11}, w_{12} vs Number of Trials

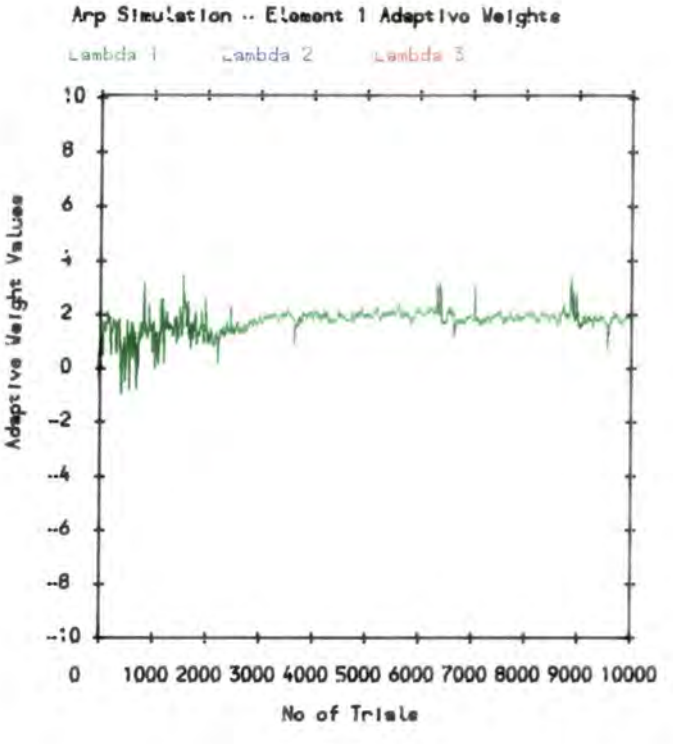


Figure 6.10c - w_{13} vs Number of Trials

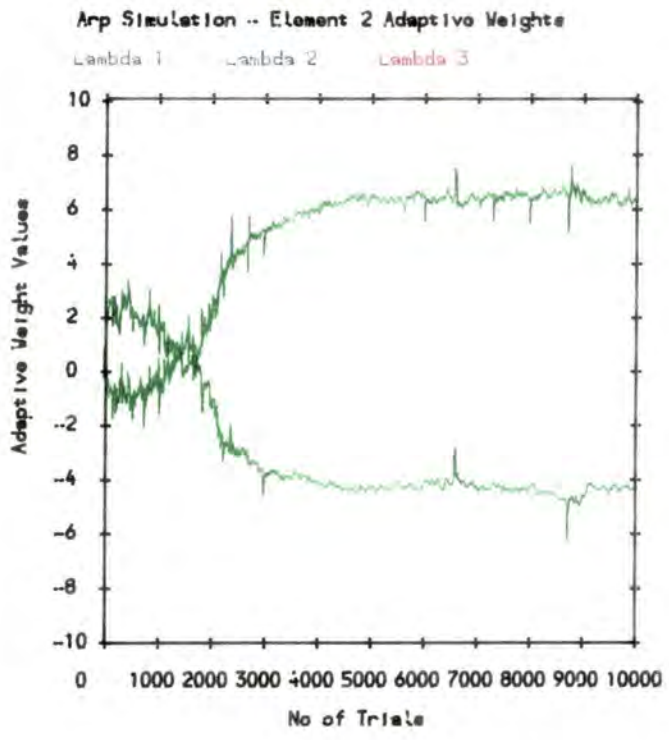


Figure 6.10d - w_{21}, w_{22} vs Number of Trials

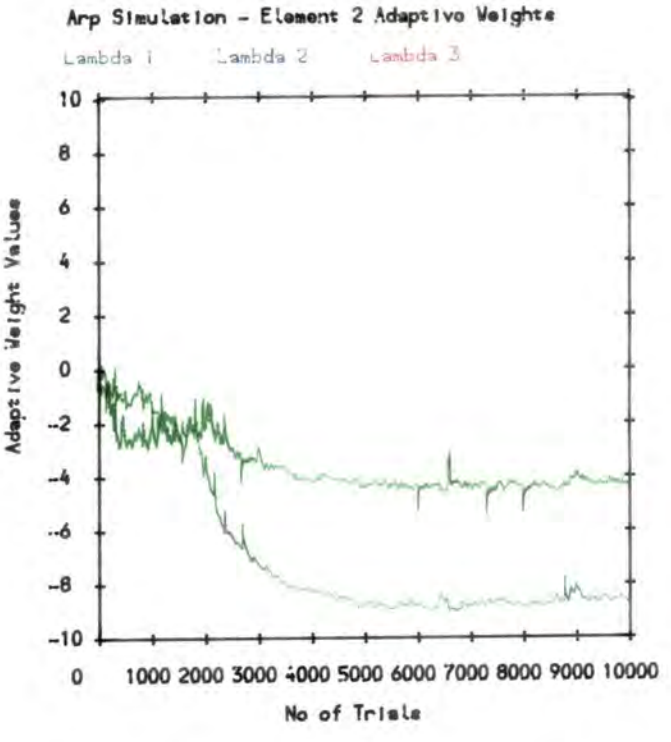


Figure 6.10e - w_{23}, w_{24} vs Number of Trials

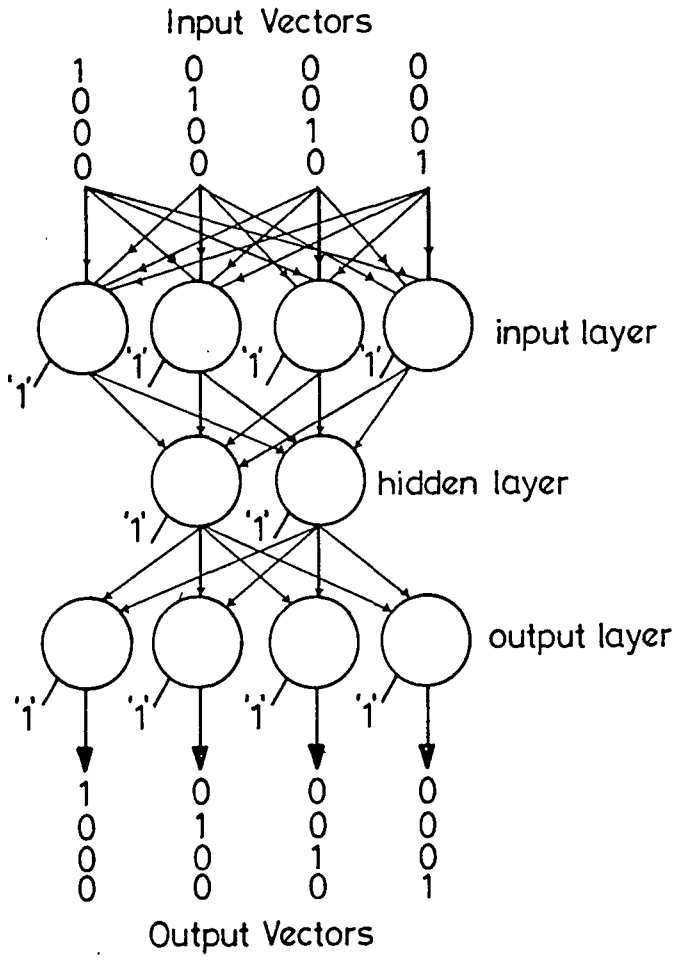
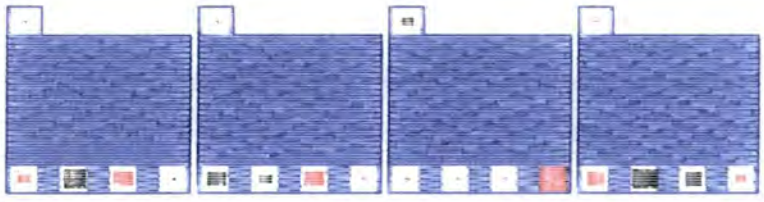
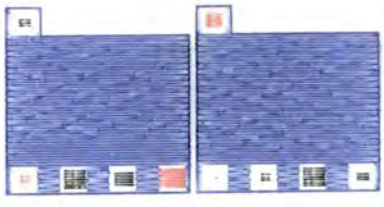


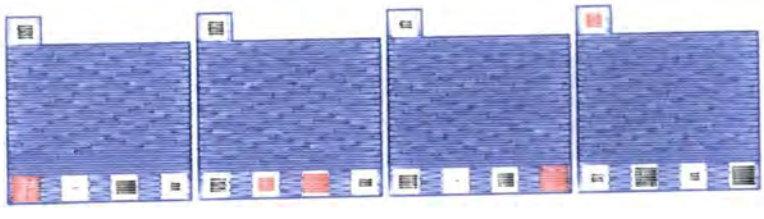
Figure 6.11 4–2–4 Encoder problem



Max Abs. Weight 0.55E 01 - Level 1



Max Abs. Weight 0.57E 01 - Level 2



Max Abs. Weight 0.70E 01 - Level 3

Figure 6.12 4–2–4 Encoder weights

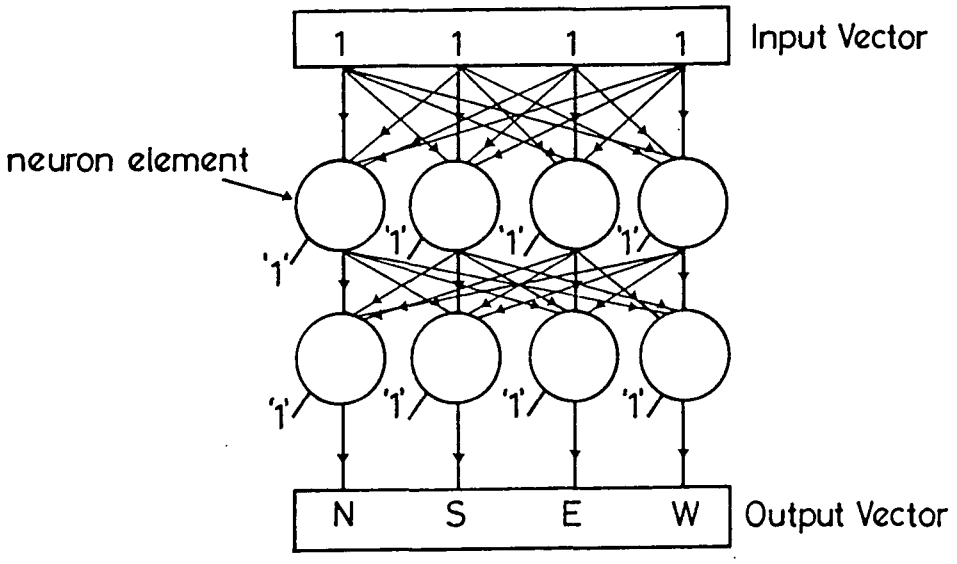


Figure 6.13 Minimum Distance array

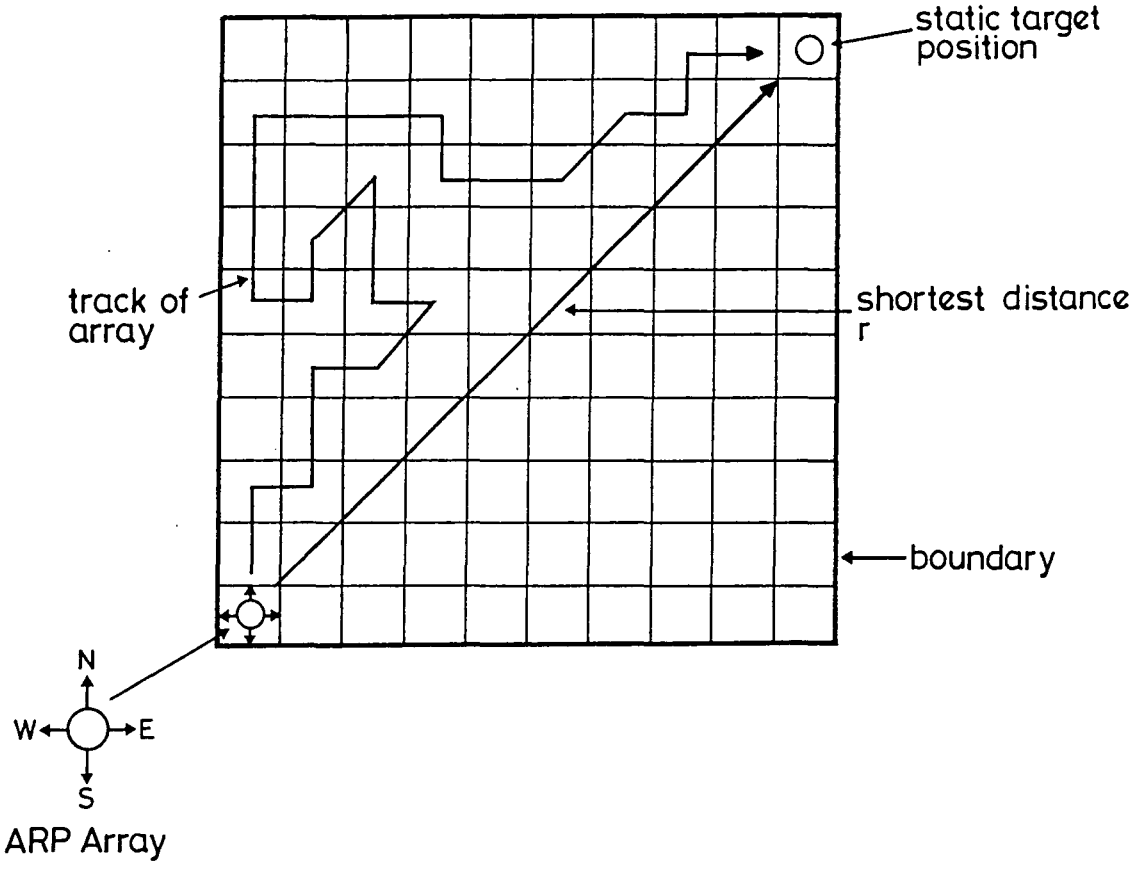


Figure 6.14 Minimum Distance environment

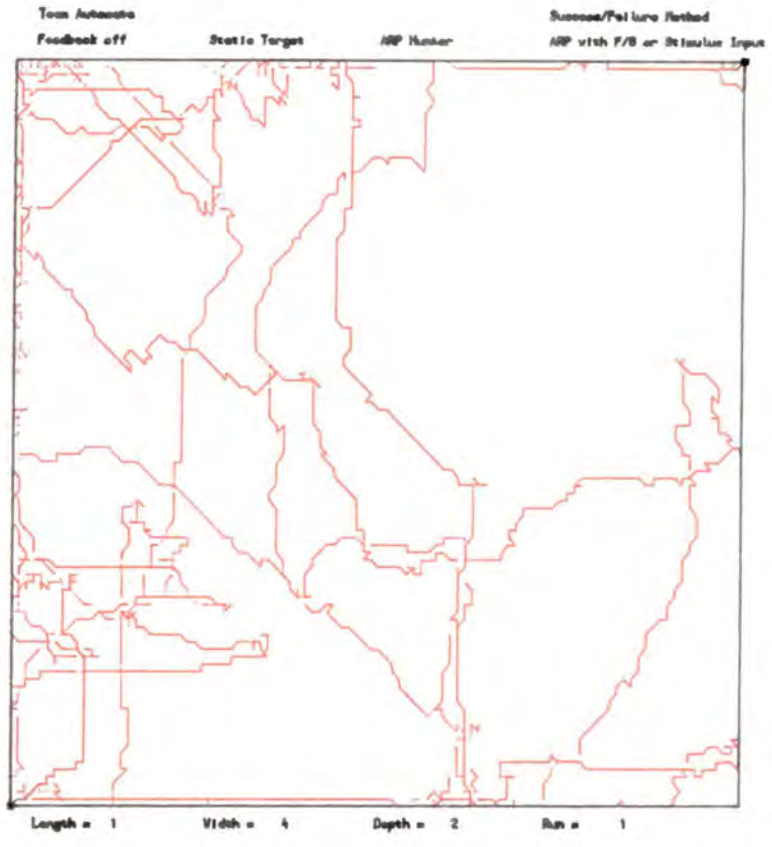


Figure 6.15a Minimum Distance problem - Success/Failure method

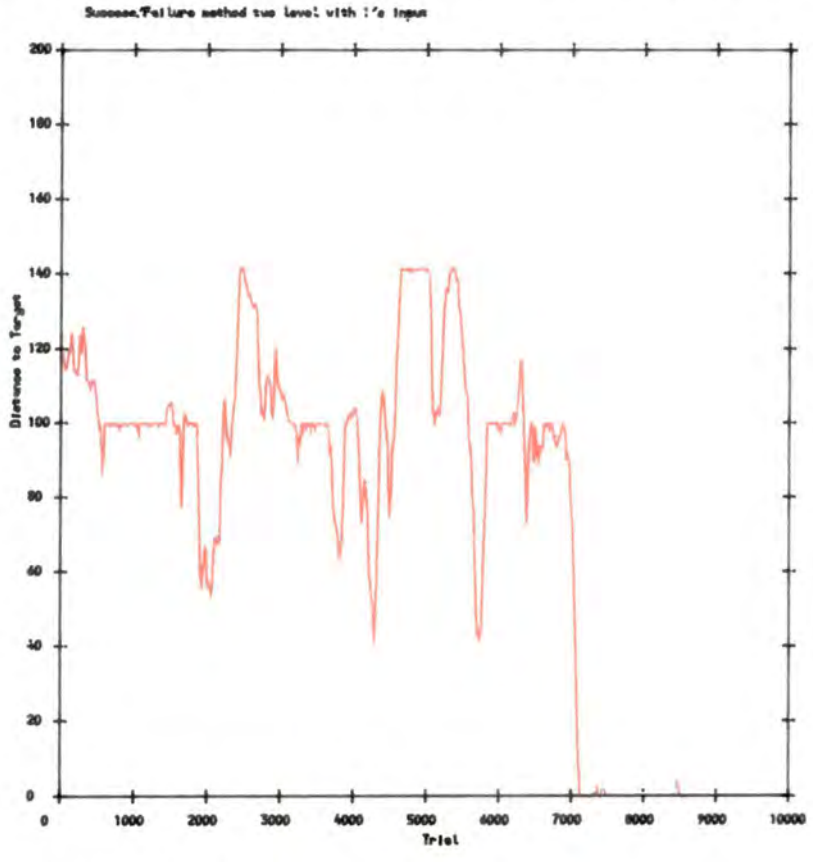


Figure 6.15b Minimum Distance problem - Success/Failure method

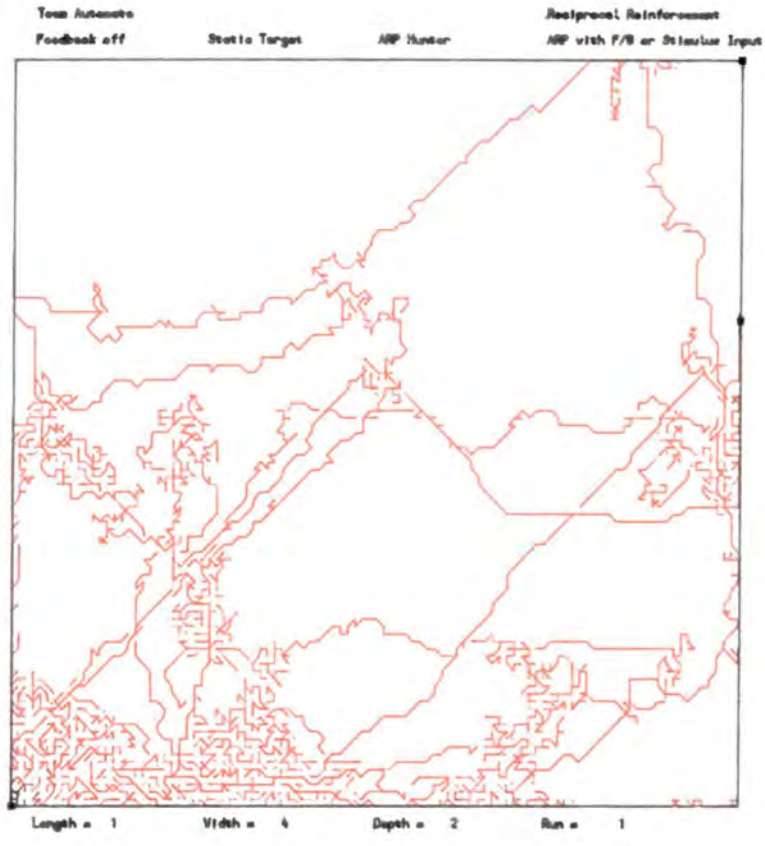


Figure 6.16a Minimum Distance problem - Reciprocal method

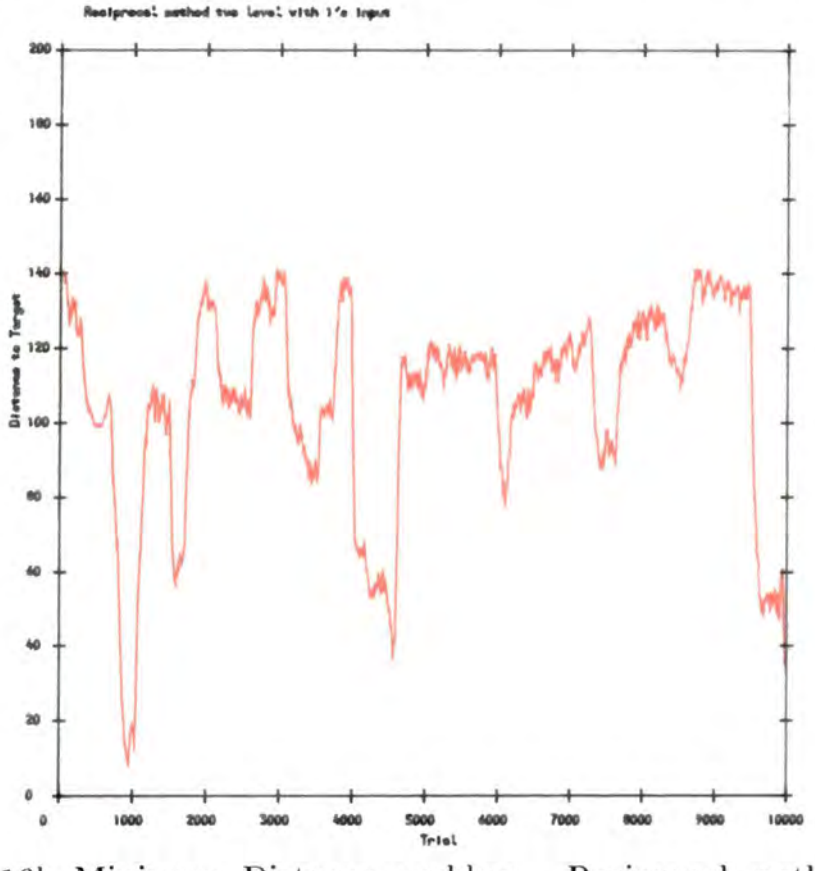


Figure 6.16b Minimum Distance problem - Reciprocal method

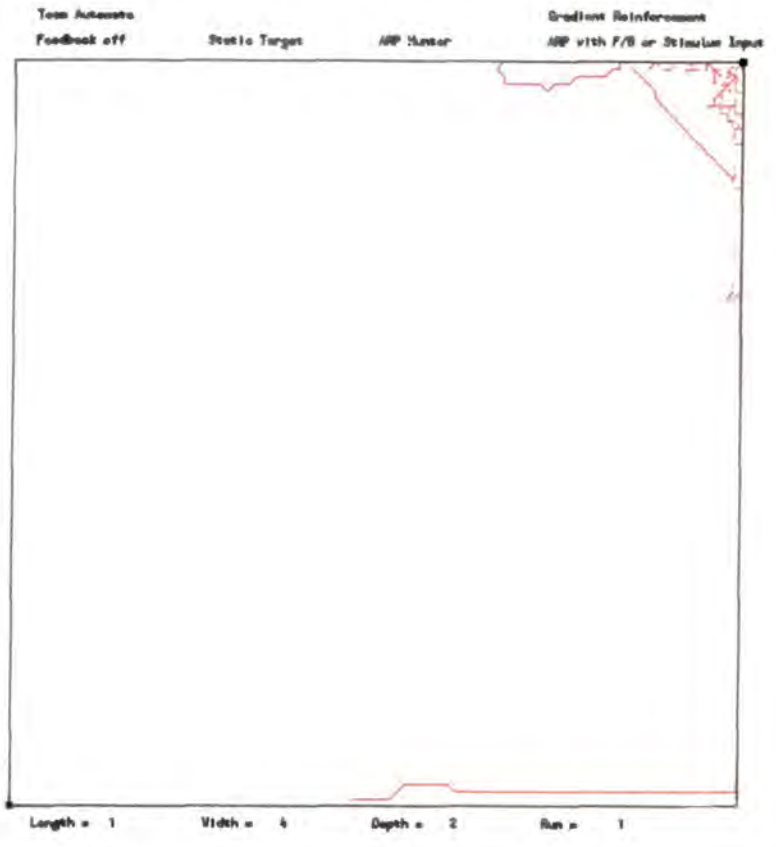


Figure 6.17a Minimum Distance problem - Gradient method

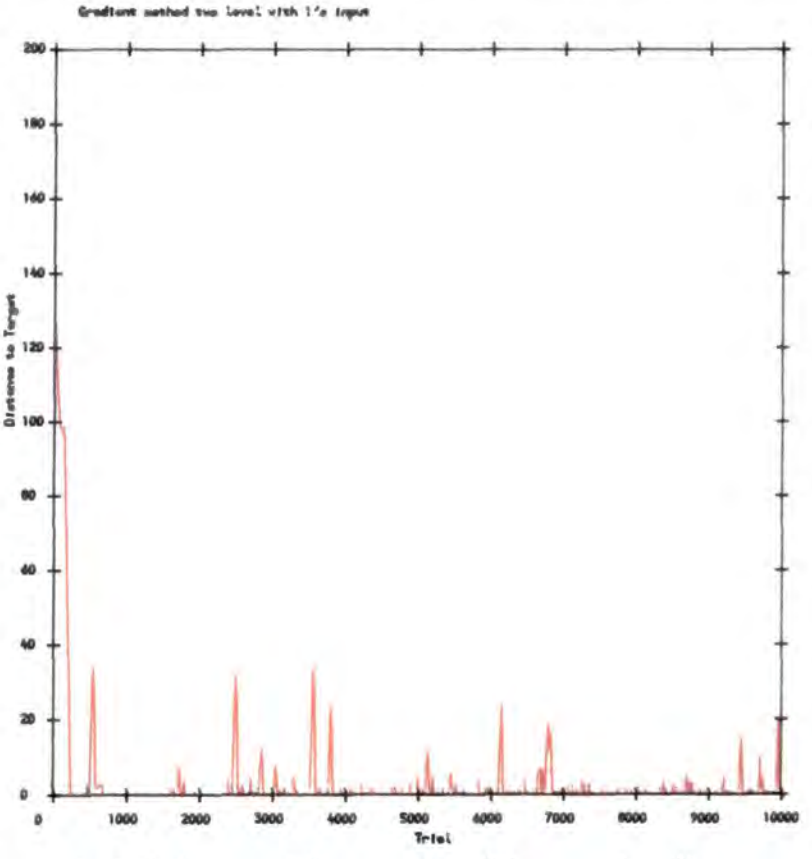


Figure 6.17b Minimum Distance problem - Gradient method

Table 6.1 - Pattern Recognition weights - Example 1

Input	Output	Weights used
0,0	0	w_t
0,1	0	$w_2 + w_t$
1,0	1	$w_1 + w_t$
1,1	0	$w_1 + w_2 + w_t$

Table 6.2 - Reward Probabilities - Example 1

x	d(x, +1)	d(x, 0)
1,0	0.1	0.9
1,1	0.9	0.1

Table 6.3 - Reward Probabilities - Example 2

x	d(x, +1)	d(x, 0)
0,0	0.1	0.9
0,1	0.9	0.1
1,0	0.9	0.1
1,1	0.1	0.9

Chapter Seven

Stochastic Implementation of Neural Networks

7.1 Introduction

This chapter describes the novel adaptation of two different types of neural network to allow the use of stochastic encoding techniques. Chapter Two introduced the concept of stochastic encoding and described how such elements use simple hardware and by their nature are noise tolerant. A key feature to note is that since the neural elements are in themselves limited to outputs between or equal to 0 or 1, no form of scaling is necessary to preserve the probability measure. It will be shown that this leads to considerable simplification since most operations then require only inversion and simple gating operations.

The first section describes a possible analogue implementation of the electronic neural element described in Chapter Five. Two methods of introducing stochastic encoding into the operation of an array of neural elements and updating the weights are then described. The chapter then examines an original stochastic implementation of the Associative-Reward Punish (A_{R-P}) element described in Chapter Six. As previously explained, the output of an A_{R-P} element is limited to either 0 or 1 and so only single bit interconnection links are necessary between elements. The advantage of implementing the internal computation of an A_{R-P} element using stochastic

methods is the resulting simplification of the hardware. Simulation results are presented for a stochastic A_{R-P} element for comparison with the examples described in the previous chapter.

The chapter then considers the deterministic error backpropagation element, and discusses a general performance criterion suitable for comparing both stochastic and deterministic feedforward networks. Simulations of the deterministic element are then presented using this criterion. A novel stochastic implementation of the error backpropagation element is then described together with a discussion of possible problems. As described in Chapter Five, the backpropagation algorithm is deterministic in operation and the element outputs lie between or equal 0 or 1. Any parallel hardware implementation (as opposed to serial computer simulation) therefore requires multibit communication links and processing. Such operations are expensive, unwieldy and biologically unviable. Despite this, the the algorithm is one of the most popular and successful for training neural network arrays. A stochastic implementation would be very useful, not only in terms of simple cheap parallel hardware with single bit interconnections between elements, but also by showing that deterministic algorithms may be coded using probabilistic techniques in this way. Furthermore, the introduction of noise through the use of stochastic coding may be beneficial to the system.

The chapter concludes by considering a potential scheme for training a stochastic A_{R-P} network using reinforcement learning, although the method is easily extended to cover the supervised learning case of stochastic error backpropagation.

7.2 Hardware Implementation of a Stochastic Neural Element

A basic electronic element was described in Chapter Five, which incorporated what was judged to be the key computational properties that might by extracted from observations of biological neurons. It was explained how the operation of some neurons appears to be analogue, but utilising some form of pulse rate coding to manipulate information between neurons. It was described in Chapter Two how stochastic processing might be considered to be a natural medium for neural nets, embodying many of the features of analogue computation in a digital, noise tolerant form.

A stochastic implementation of the neural element of Chapter Five could be formed from a hybrid arrangement of analogue and digital components. Figure 7.1 depicts a potential implementation of such an element and is intended to represent the key components required rather than form a complete schematic. It shows an operational amplifier which is configured to form the summation of the voltages on its inputs in the standard manner. The input resistors represent the weights and are dynamically implemented using voltage controlled resistances such as JFET devices. Positive and negative weights are achieved by using two lines, one carrying a positive going pulse and the other carrying a negative going pulse, generated simultaneously by the previous element and input, via the connection weight 'resistors' to the inverting input of the op-amp. The output of the op-amp is stochastically thresholded using a comparator with a reference voltage generated from noise. The output pulses, one positive and one negative are generated on both output lines simultaneously, when the op-amp output exceeds the noise reference.

The amplifier is required to have a saturating output characteristic as described in Chapter Five, where the probability of the element producing a pulse, given input x, is given by;

$$p(t) = \frac{1}{1 + e^{-s(t)/T}} \tag{7.1}$$

The resistor–capacitor combination on the input is included to shift the characteristic from a step function to this curve corresponding to a 'Temperature' change T.

The weight updates are introduced using stochastic pulses, input to the low pass filter (ie. conversion of the stochastic pulse sequence to a deterministic voltage proportional to the generating probability of the sequence) which, for an appropriate choice of time constant results in a time decaying bias voltage, which is 'refreshed' or increased by the arrival of weight update pulses and which controls the dynamic resistances. In this way, the weights are preserved over a limited interval, the weight updating process being used to set and maintain the weights. Synchronous operation of the array may be accomplished using a clock reference although it would be interesting to evaluate the asynchronous performance of such an array since this appears to be the mechanism adopted by the biological networks in nature.

7.3 Training a Stochastic Network

It is first necessary to define some of the terms to be used which describe how the training vector set is presented to a stochastic neural network. A '**run**' is defined as a single execution of the simulation program

when training an array from either initially zero or randomised weights. A 'pass' is defined as a sufficient number of trials during a run to allow random presentation of the training set to the array. This was usually taken to be twice the number of input–output vectors in the set. A **'batch'** is defined as the repeated presentation of a single input–output vector during which time the weights may be updated on each trial or the difference values may be accumulated until the completion of the batch and then weights adjusted. A **'sweep'** is defined as one or more passes during which time, the required difference values, which may be weight updates or element errors are accumulated. Only when the sweep is completed, are the weights updated. The batch process may itself be part of a sweep, so that when an input-output vector is randomly selected for presentation, it is presented a number of times consecutively.

Given these definitions, there are then two potential methods which may be identified for use in stochastic training of a network. The first involves forming the static average of the stochastic weight updating pulses over a batch or a sweep. This allows an estimate of the generating probability representing the weight update to be formed. This method is useful as a means of testing stochastic encoding and checking that the algorithm is working as expected. The second method involves stochastically updating the weights on each trial over a pass without batching. This requires that a moving accumulation or some average measure, is taken over a suitable interval of the update pulses received. This value is then used as the weight for the next trial and so on and corresponds to the low pass filter concept

as described in the last section.

These two methods raise the question as to how accurate these estimates should be in order to ensure that the algorithm operates as it would, were the method still deterministic. It is expected though, that the additional noise introduced by stochastic encoding will be beneficial to the process, discouraging the algorithm from becoming trapped in local minima.

7.4 Stochastic Implementation of an A_{R-P} element

This section considers the stochastic implementation of the A_{R-P} element and examines the weight updating algorithm. Taking the basic hardware element as described in the earlier section, the area which requires further examination is the conversion of the weight updating scheme to permit stochastic encoding. Recalling the A_{R-P} element weight updating rule, described in Chapter Six (equation 6.21);

$$\Delta w_n(t) = \begin{cases} \rho[2y(t) - 1 - E\{a(t) \mid s(t)\}]x_n(t) & \text{if Success} \\ \rho\lambda[1 - 2y(t) - E\{a(t) \mid s(t)\}]x_n(t) & \text{if Failure} \end{cases} \tag{7.2}$$

It was shown that the expectation of the automaton state $E\{a(t) \mid s(t)\}$ could be expressed as

$$\begin{aligned} E\{a(t) \mid s(t)\} &= -1.p_t^{-1x} + 1.p_t^{+1x} \\ &= 2p_t^{+1x} - 1 \end{aligned} \tag{7.3}$$

It may be seen that for $\rho = 0.5$, equation 7.2 reduces to

$$\Delta w_n(t) = \begin{cases} [y(t) - p_t^{+1x}]x_n(t) & \text{if Success} \\ \lambda[1 - y(t) - p_t^{+1x})]x_n(t) & \text{if Failure} \end{cases} \tag{7.4}$$

By noting that $y(t)$ can only take on the values 0 or 1, the two weight updating cases are; For $y(t) = 0$

$$\Delta w_n(t) = \begin{cases} [-p_t^{+1x}]x_n(t) & \text{if Success} \\ \lambda[1 - p_t^{+1x})]x_n(t) & \text{if Failure} \end{cases} \tag{7.5}$$

and for $y(t) = 1$

$$\Delta w_n(t) = \begin{cases} [1 - p_t^{+1x}]x_n(t) & \text{if Success} \\ \lambda[-p_t^{+1x})]x_n(t) & \text{if Failure} \end{cases} \tag{7.6}$$

Consider the physical meaning of equations 7.5 and 7.6. The weights are changed by a factor dependent on the probability of firing. In the simulations of Chapter Six, this probability was expressed internally as a deterministic value formed from a saturating weighted summation. This may well be a plausible mechanism in the biological neuron as it requires simple local analogue operations.

An alternative way of approaching this method is to examine the feasibility of using a stochastic sequence representing this probability of firing. Clearly the element output $y(t)$ itself is such a sequence. However this cannot be used, since sequence independence needs to be preserved, as explained in Chapter Two. Instead, an appropriate stochastic sequence may be obtained by independently re–thresholding the output of the neuron for use as an internal sequence. This might be done several times between trials, ie. a batch mode, with the original output of the neuron $y(t)$ and the reinforcement r(t) being held constant during this time, the weight updates being accumulated before the weights were changed.

By using this method, for the arrangement of Figure 7.1, each weight may be represented using the DLSR representation as two moving accumulation counters, one for a positive weight and one for a negative weight. The length of the shift register adjusts the time constant for the weight to decay. It is also possible to introduce λ as a stochastic sequence gating (ie. multiplying)

the neuron internal probability sequence where appropriate. The absence of λ is simply represented by a gating sequence which is always on. The problem here is that if λ is small, e.g. $\lambda = 0.01$, then this value is conveyed by a sequence averaging one pulse in one hundred. For this reason, it was considered better to adjust the counter increment scale using λ rather than use a multiplying sequence.

Note also that the probability sequence $(1 - p_t^{+1x})$ is simply the inversion of p_t^{+1x}. The probability sequence $-p_t^{+1x}$ is simply the sequence p_t^{+1x} but conveyed along the DLSR negative line. Figure 7.2 shows the stochastic weight updating scheme for the A_{R-P} algorithm.

7.4.1 Stochastic A_{R-P} Simulations - One Element

In order to examine the performance of a stochastic A_{R-P} element, selected simulations from Chapter Six were repeated. The element operation was that as described by equation 7.4. The weight updates were computed stochastically and used to update the total weight on each trial. This is equivalent to a counter mechanism with a shift register longer than the total number of trials ie. with no weight decay. A batch mechanism was included over which an estimate of p_t^{+1x} could be made from a short average of the element output taken over the batch with the input held, before updating the weights. Normally, for a purely stochastic update mechanism as shown in Figure 7.2, the batch length would be 1, however this mechanism was included both as a comparison and a test facility.

The stochastic A_{R-P} element was simulated for the arrangement of Figure 7.3 with reward probabilities as shown in Table 7.1 for three different values of λ, with $\rho = 0.5$ and the system 'Temperature', T = 0.5.

$$\lambda_1 = 0.01$$

$$\lambda_2 = 0.05 \qquad (7.7)$$

$$\lambda_3 = 0.25$$

Figure 7.4a shows the performance M_t of the element, M_t computed as before. M_t was averaged over 100 sequences of 10,000 trials. In this simulation, the estimate of p_i^{+1x} was taken over a batch length of 1, ie. the estimate of p_i^{+1x} was purely stochastic. Figure 7.4b shows the values of the element weights, w_1, w_2 over the trials for the three values of λ. Figures 7.4c and 7.4d show M_t over 1 sequence of 10,000 trials. The dotted lines again represent the theoretical asymptotes as calculated using the procedure described in Appendix Six.

The results may be interpreted as follows. Firstly, consider Figures 7.4a and 7.4b. Although the performance M_t converges to the expected value of 0.9 it will be seen that the weights do not converge to the predicted asymptotes. This is because the stochastic weight update taken over a batch of 1 allows only coarse discrete values to be used for the update. This is immediately visible from Figure 7.4d which shows the weights over a single run with their characteristic 'jagged' appearance. This may be contrasted with the subsequent repeat of the simulations, this time using a batch length of 10 which permits a better estimate of p_i^{+1x} to be made. Again, Figure 7.5a shows the performance M_t of the element, with M_t averaged over 100

sequences of 10,000 trials. Figure 7.5b shows the values of the element weights, w_1, w_2 over the trials for the three values of λ and Figures 7.5c and 7.5d show M_t over 1 sequence of 10,000 trials. In this case, the averaged weight values converge much closer to the predicted values as shown in Figure 7.5b and the weights on each run are less 'spiky' and coarse.

7.4.2 Stochastic A_{R-P} Simulations – Two Elements

The simulations were repeated, this time for two A_{R-P} elements in a linear configuration as shown in Figure 7.6 for the Reward Probabilities shown in Table 7.1. Figure 7.7a shows the performance M_t of the elements with M_t taken over 1 sequence of 10,000 trials. For this simulation, the estimate of p_t^{+1x} was again taken over a batch length of 1 so the estimate of p_t^{+1x} was stochastic. Figures 7.7b and 7.7c show the values of the element weights, w_{11}, w_{12}, and w_{21}, w_{22} over the trials for the three values of λ.

This was repeated for a batch length of 10. Figure 7.8a shows the performance M_t of the elements with M_t taken over 1 sequence of 10,000 trials. Figures 7.8b and 7.8c show the values of the element weights, w_{11}, w_{21}, and w_{21}, w_{22} over the trials for the three values of λ. These results are similar to the previous single stochastic element simulation. The array converges to the expected performance level of 0.9 quickly for the lowest value of λ as in the previous deterministic simulations. The weights for the case taken over a batch length of 1 are much coarser than the relatively smooth weights taken over a batch length of 10.

7.4.3 A_{R-P} Simulations – Exclusive OR Problem

For the third test of the stochastic element, the Exclusive OR problem was repeated. The Reward Probabilities are shown in Table 7.2 and the 2 element A_{R-P} configuration in Figure 7.9. As before, both elements were presented with the input vectors, but element 2 additionally has the output of element 1 as an input and both elements had a threshold input. Element 1 adjusts three weights while Element 2 is adjusting four weights. If the two stochastic A_{R-P} elements learn to cooperate, a maximum performance of 0.9 can be obtained.

The results of the simulation are shown in Figures 7.10(a-e), taken over one sequence, using $\lambda = 0.01$ and $\rho = 0.5$ and batch length 1 (ie. purely stochastic updates). The plots show that the performance asymptotes to the optimum value indicating that the elements have correctly learned. As before, the weight values are coarse due to the stochastic updating. The simulations were repeated for batch length 10 and shown in Figures 7.11(a-e). Again the performance asymptotes to the optimum value.

7.5 Stochastic Implementation of Error Backpropagation

The stochastic A_{R-P} implementation described in the last section showed a simple way of performing the weight updating operations for a stochastic element. One of the most popular methods for training feedforward neural networks involves error backpropagation and the architecture of a single element was described in Chapter Five. Figures 7.12a and 7.12b show both the final layer and previous layer elements. In similar fashion to the A_{R-P}

method previously described, adaptation to a stochastic architecture can take on three forms.

The first step involves encoding all the elements inputs and outputs, which are deterministic, to a stochastic form using the standard technique of comparison against a random (ie. noisy) threshold. By observing the outputs of each of the elements in the previous layer of the network over a batch, then static averages of these values over this period will yield an estimate of the deterministic value represented by y_i for all j elements. Similarly, estimates may be obtained for error quantities (for final layer elements) and $\frac{\delta E}{\delta s}$ values from lower levels (for the remainder of the elements in the network). Local values are available to the element in a deterministic form, such as its output and the $\frac{\delta E}{\delta s}$ quantity before stochastic encoding and output. It should then be possible to perform error backpropagation in the standard way.

This method is not very helpful as, although it introduces single bit interconnection, the processing involved at each element remains deterministic. Instead, stochastic encoding techniques may be applied, reducing Figures 7.12a and 7.12b to Figures 7.13a and 7.13b by utilising the properties of stochastic encoding for the required arithmetic operations. One potential problem lies in the possibility that that some form of scaling might be necessary to permit the weight updating process to function when the array is close to the solution, since the return error signals would be small and in stochastic terms, infrequent.

An important point to note from these diagrams is that sequence independence, which is very important in stochastic computation, is virtually

guaranteed by the weighted summation and re-encoding process at each neuron. Furthermore, consider the updating of the weights. In the stochastic case, it is only the presence or absence of an input event together with an error feedback pulse which affects the update. This may have some parallels with Hebbian correlation, [87], as explained in Chapter Five. Note also that the outputs of each of the elements are stochastically thresholded more than once to provide local independent sequences for some of the mathematical operations, particularly for the final level neurons which do not have the weighted summation isolation during the reverse pass. The operation of the array can then proceed as follows. The weight updates can either be accumulated over a sweep or over a series of batches during a sweep and then adjust the weights, or the weights can be adjusted stochastically during each trial. Note also how the function Y(1-Y) has maximum value 0.25. In the stochastic implementation, this function is scaled up by a factor of 4 to make full use of the probability field available, with the weight update being adjusted appropriately.

7.5.1 Performance Criterion for Feedforward Networks

The question now arises, how to judge the performance of a feedforward network in order to gauge when it has learned. The earlier performance criterion used to analyse the performance of simple A_{R-P} networks was based on the expectation of reinforcement, given the probability of the state of each element and is only suitable for reinforcement methods. A more general measure was therefore needed which could cope with all types of feedforward

network.

Recall equation 5.14 in Chapter Five which gave an expression for the total error at the output of a network, based on the sum of the absolute difference between the output of the network and the required output for the particular input.

$$E = \frac{1}{2} \sum_{c} \sum_{j} (y_{j,c} - d_{j,c})^2 \tag{7.8}$$

For deterministic backpropagation, this will never be exactly zero for complete learning since the element outputs asymptote to their limits of 0 and 1 due to the exponential term and more commonly stabilise at values near 0.9 and 0.1. Applying the same criterion to the actions of an A_{R-P} network results in a slightly different problem. In this case, the probability of element output asymptotes to the limits of 0 and 1 and so the elements tend to stabilise at values where the probability of firing is near 0.9 and 0.1. In this way, the element weights may be such that, for instance, the probability that the element remains dormant is high but there still remains a small chance that it will fire.

The simulation programs used for the following results examined the output error at each trial. The local error at each output element was corrected to zero if appropriate, allowing for the asymptotic outputs. If the total resulting error was zero then the value one was input to an accumulator, otherwise a zero was input. The moving average taken over 500 trials yielded a percentage correct result. If this was higher than 90%, then the array was judged to have learned correctly. Whilst this was very much an *ad hoc* arrangement, it was found to be satisfactory in dealing with

performance comparisons of both A_{R-P} networks and error backpropagation networks although it was considered only suitable for a small input/output training set when taking the average over so few trials to allow full random presentation of the training set.

7.5.2 Error Backpropagation Simulation – one element

For comparison purposes, the deterministic error backpropagation algorithm was simulated for the element shown in Figure 7.14. The element was presented with a single input plus threshold during the forward pass. After the element had produced an output, an error signal was returned by the Environment, computed as in equation 7.8 corresponding to the truth table shown in Table 7.3. The change in weights were accumulated over a sweep containing one pass. At the end of the sweep, the weights were updated. The performance (% correct over 500 trial window) is shown in Figure 7.15a for three different values of ϵ, with the 'momentum' term $\alpha = 0.5$ and the system 'Temperature', T = 0.5.

$$\begin{aligned} \epsilon_1 &= 0.1 \\ \epsilon_2 &= 0.25 \\ \epsilon_3 &= 0.5 \end{aligned} \tag{7.9}$$

The weights for the element are shown in Figure 7.15b and exhibit a smooth convergence. Clearly the fastest learning occurs for $\epsilon = 0.5$.

The stochastic implementation of the error backpropagation element was then simulated for the same problem and parameters. In this simulation, the weight update pulses were averaged over n=1 trials and the weights

updated every 2n trials. This method was adopted rather than updating over a sweep to allow flexibility in the interval for averaging the weight update. Figure 7.16a shows the performance (% correct over 500 trial window) and the weights are shown in Figure 7.16b. These results may be compared with the deterministic case and interpreted as follows. The elements performance quickly climbs, though tailing off at the asymptote to the 100% correct level with the fastest learning occurring again for $\epsilon = 0.5$. The most interesting feature appears in the weight plot of Figure 7.16b. Recall how, as the performance increases, the error signal becomes smaller and in stochastic encoding, infrequent. Note how the weight corrections consist of quantised steps (due to the stochastic updating, with the update pulses averaged over n=1 trial and weights updated every 2n trials). It will be seen that the changes in weights become less and less frequent as the problem becomes solved, corresponding to fewer and fewer error pulses.

The simulation was repeated, this time averaging the update pulses over n=10 trials with the weights updated every 2n trials. Figure 7.17a shows the performance (% correct over 500 trial window) and the weights are shown in Figure 7.17b. In this case, the elements performance appears slower although the best learning is exhibited by $\epsilon = 0.5$. Note how the weights are smoother due to the limited averaging before updating.

7.5.3 Error Backpropagation Simulation – two elements

The simulations were then extended to cover two deterministic elements with the topology shown in Figure 7.18 for the truth table shown in Table

7.3 and parameters as before. Figure 7.19a shows the array performance and Figures 7.19b and 7.19c show element one and two's weights respectively. The results show a fast learning rate, though not quite as fast as the single element case. The weights asymptote smoothly to a steady state.

The simulations were repeated using stochastic error backpropagation elements using the same procedure as before. The performance for the elements, with the weight updates averaged over n=1 trials and updated every 2n trials is shown in Figure 7.20a with Figures 7.19b and 7.19c showing element one and two's weights respectively. These results are particularly interesting. Note the typical stochastic climb of the performance when the two elements must cooperate in order to minimise the error as shown in Figure 7.20a. Consider Figure 7.20c which shows the output element weights. The weights are clearly changing more rapidly than the input element weights shown in Figure 7.20b. This is because the output element directly receives the error signal from the Environment which is used to control the weight update. The input element receives an error signal from the output element based on this Environmental signal but also based on a function of the output elements state. Some degree of probability attenuation inevitably takes place and the result is that the element weights change at a lower rate.

Repeating the simulations and averaging the updates over n=10 trials and updating the weights every 2n trials as before yields the performance results shown in Figure 7.21a and weights for elements one and two shown in Figures 7.21b and 7.21c respectively. It may be seen that the elements performance is considerably more sluggish with the fastest climb exhibited by

$\epsilon = 0.5$.

7.5.4 Error Backpropagation Simulation – zero recognition

It was then decided to see how the stochastic error backpropagation performance was able to cope with a large number of elements. The 5 x 5 x 2 zero recognition problem as described in Chapter Five was used, the topology is shown in Figure 7.22. Since the array consists of 50 elements, it was considered that weight plots would not serve any useful purpose and so only performance plots have been included. Figure 7.23a shows the performance of the deterministic element array. Figure 7.23b shows the performance of the stochastic element array with updates averaged over n=1 trials and weights updated every 2n trials and Figure 7.23c shows the performance of the stochastic element array with updates averaged over n=10 trials and updated every 2n trials. Clearly the deterministic array learns to solve the problem extremely quickly. The stochastic array performance as shown in Figure 7.23b is also very quick although a slower performance is exhibited by the array described in Figure 7.23c. Again, the fastest performance is given by the use of $\epsilon = 0.5$.

7.5.5 Error Backpropagation Simulation – Exclusive OR problem

The final simulations were performed for the topology shown in Figure 7.24 for the Exclusive OR problem whose truth table is shown in Table 7.4. Since each element has three weights and there are four elements, it was considered that to show plots of all of these for all of the parameters would

not be practical and so only limited weight plots are included. Figure 7.25a shows the performance of the deterministic array with all but the $\epsilon = 0.1$ array solving this non-trivial problem. Figures 7.25b and 7.25c show some of the weights in the array.

This simulation was repeated for the stochastic array, with weight updates averaged over n=1 trials and weights updated every 2n trials. Figure 7.26a shows the performance of the array with all values of ϵ enabling the array to solve the problem. In this simulation, approximately 90% performance was obtained by using $\epsilon = 0.25$ although all values gave better than 80% performance. Figures 7.26b and 7.26c show some of the weights in the array.

This simulation was finally repeated for the stochastic array, with weight updates averaged over n=10 trials and updated every 2n trials. Figure 7.27a shows the performance of the array which in this case has not solved the problem. Figures 7.27b and 7.27c show some of the weights in the array which are only slowly climbing and converging.

In conclusion, it appears that stochastic error backpropagation is feasible and that direct stochastic updating of the weights is not only possible but preferable to longer averaging intervals.

7.6 Training a reinforcement learning neural network

The discussions so far have examined the adaption of weights in a learning array by response to an environmental reinforcement signal but the source of this signal has not been described for a practical implementation. Figure 7.28 shows a possible scheme for training a reinforcement learning

neural network. It involves simply placing the network in parallel to the logic elements representing the function to be learned. If a randomly generated series of binary signals is fed to both, then an Exclusive NOR check will result in a reinforcement signal being produced for signals in agreement.

It will be noted that this method is deterministic, ie. a signal is only issued for a correct result. Only minor modifications are necessary to introduce a probabilistic response, by simply using this signal to gate a stochastic reinforcement sequence representing the reward probability, the absence of the signal enabling a reinforcement sequence representing the punish probability. This probabilistic reinforcement sequence then becomes the signal fed back to the learning system.

This signal should then enable the neural network to converge to a solution using the appropriate learning methods. The method is easily adapted to provide local error signals for stochastic error backpropagation. This technique is perhaps more familiar from adaptive control systems where the system adapts to follow a reference model.

7.7 Future Work

Future work should include a comprehensive study of a stochastic error backpropagation network using different parameters in order to fully evaluate the method. It would be very interesting to construct a simple element network using op-amps as described and examine both stochastic error backpropagation and also A_{R-P} methods of reinforcement for small problems.

7.8 Conclusions and Summary – Chapter Seven

A new performance criterion has been introduced which has been used to compare small error backpropagation and A_{R-P} networks. Preliminary simulation results have been presented for novel stochastic error backpropagation and stochastic A_{R-P} arrays.

Whilst most current research in neural networks is applied to fast simulation techniques (including custom hardware) rather than actual neuron–like hardware, this chapter has shown that two main neural network training methods may be reduced to a simple architecture by using stochastic encoding techniques and that this may result in cheap, easily expandable hardware arrays. The combination of analogue and stochastic techniques has particular advantages, especially the large fan–in & fan–out, or convergence and divergence which is a major characteristic of connectionist systems.

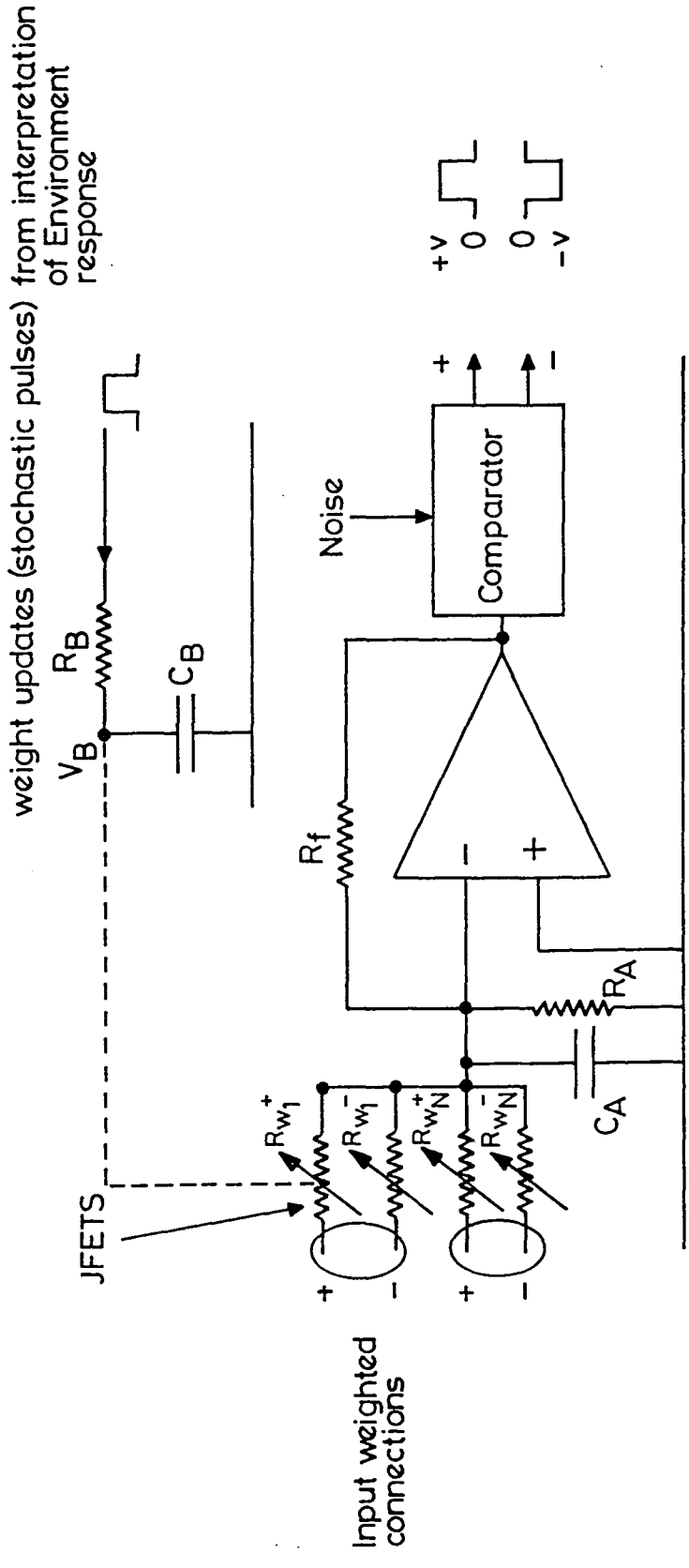
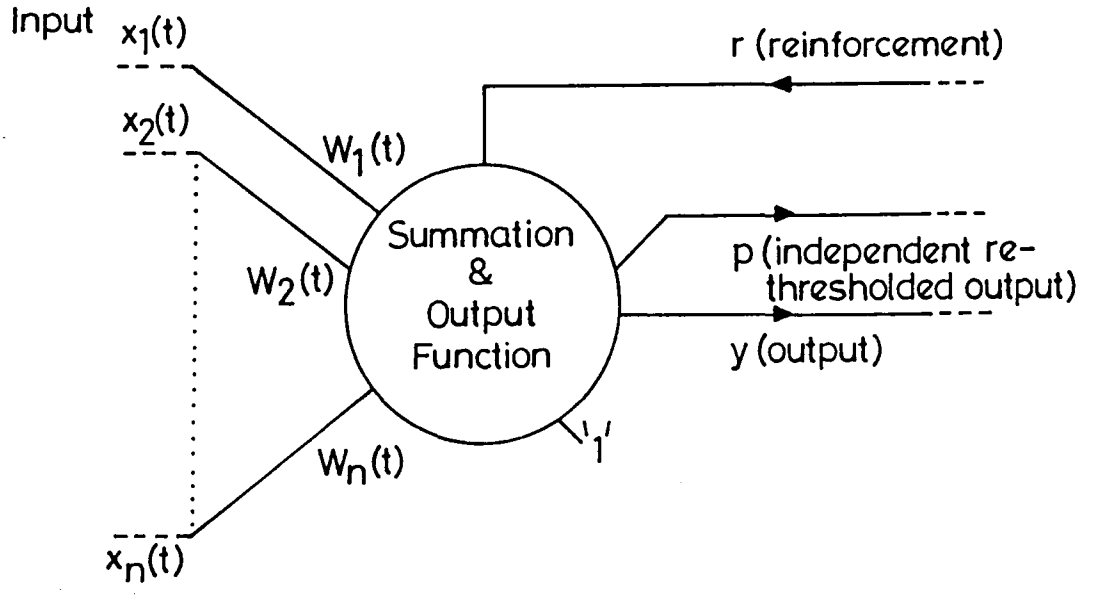


Figure 7.1 Hardware Implementation of a Neural Element



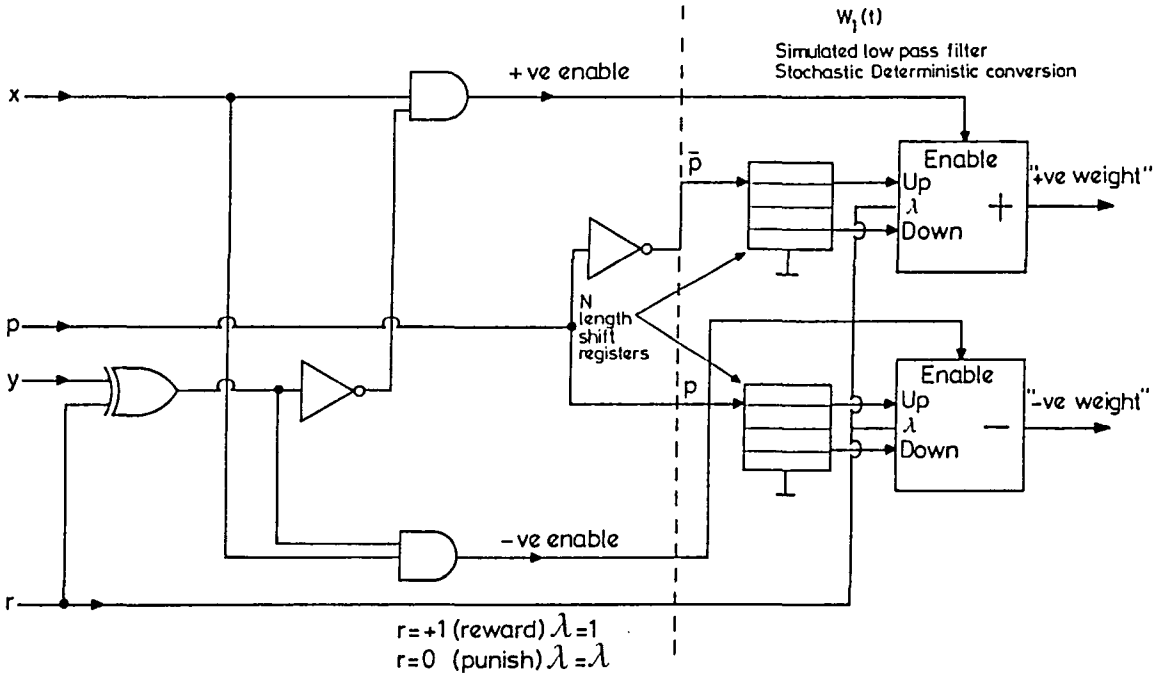


Figure 7.2 Stochastic Implementation of the A_{R-P} algorithm

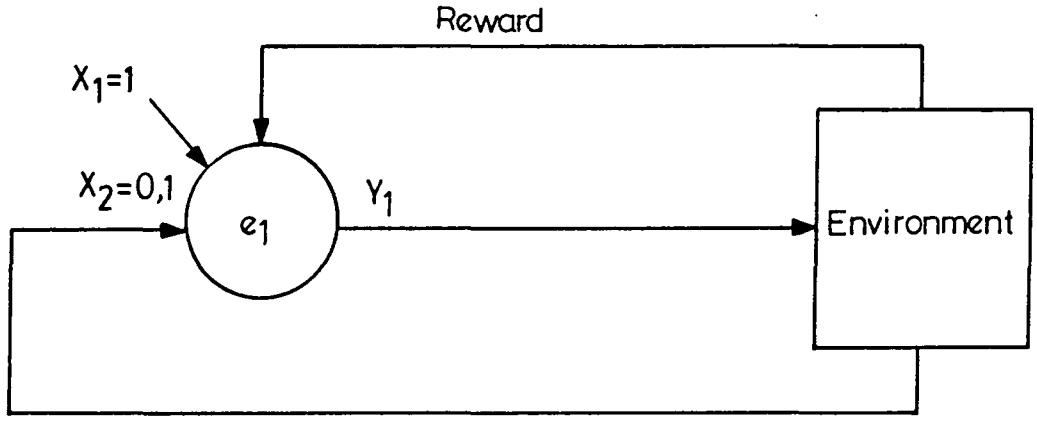


Figure 7.3 Automaton, Input and Environment

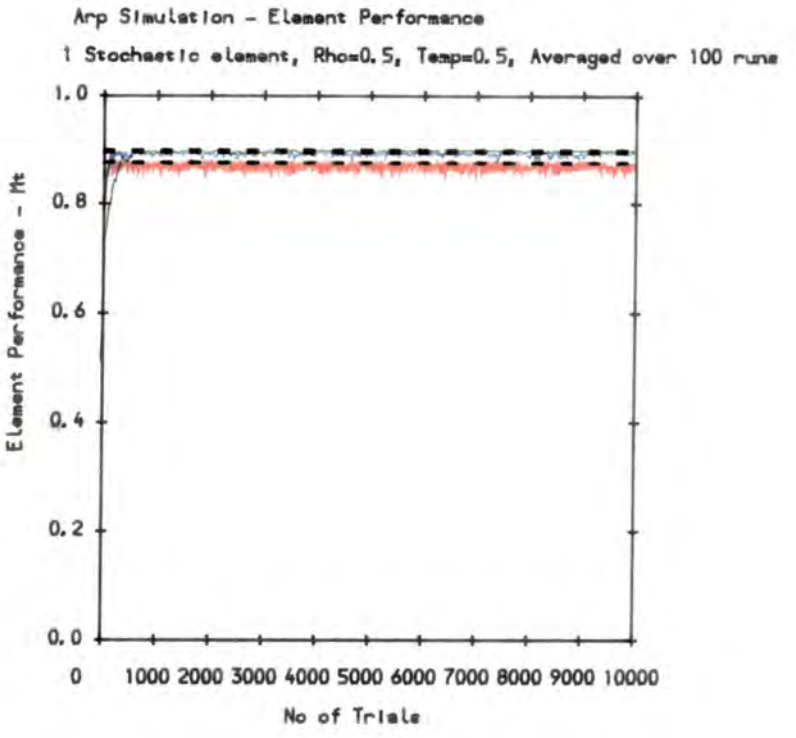


Figure 7.4a - M_t vs Number of Trials (Batch 1)

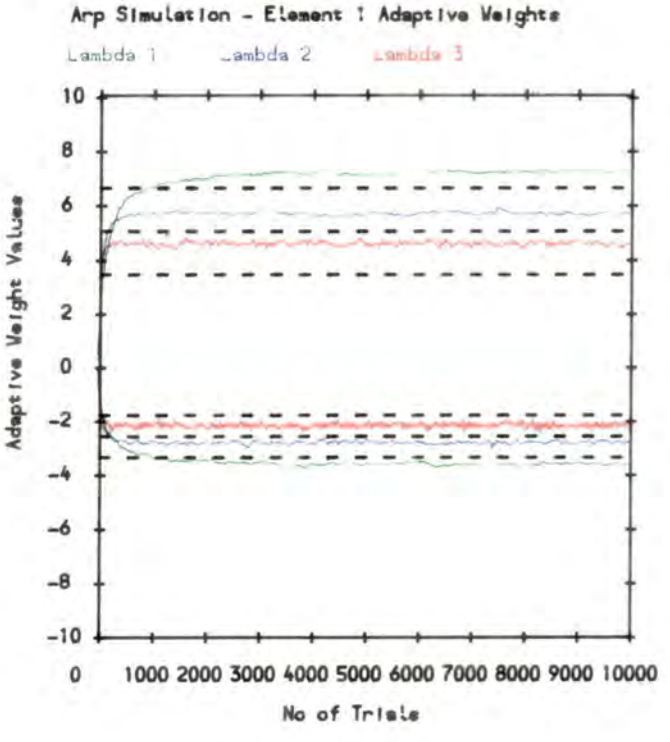


Figure 7.4b - w_1, w_2 vs Number of Trials (Batch 1)

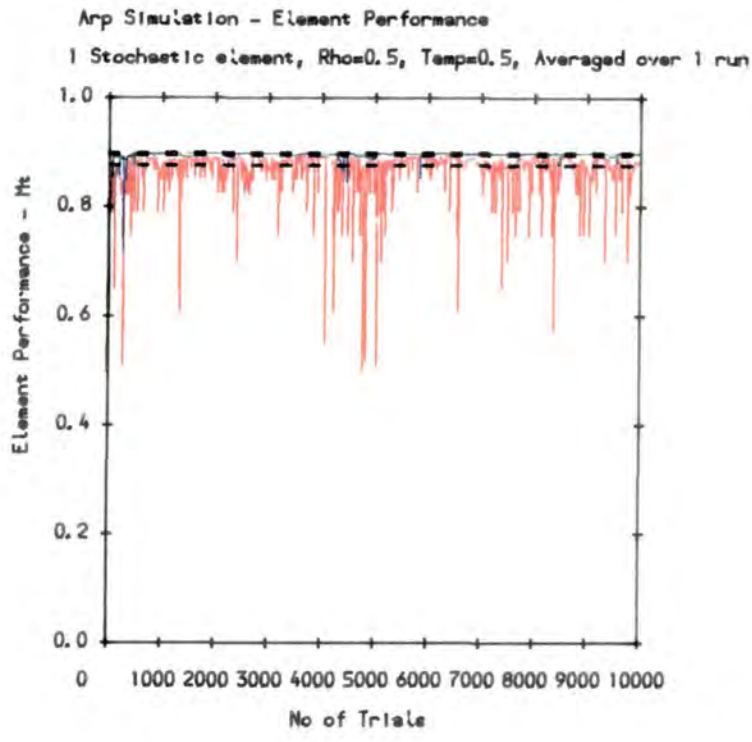


Figure 7.4c - M_t vs Number of Trials (Batch 1)

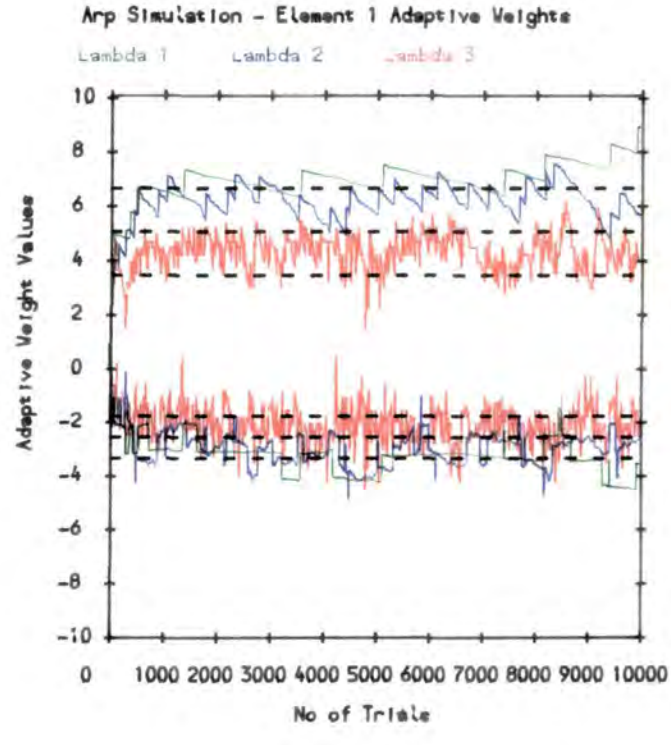


Figure 7.4d - w_1, w_2 vs Number of Trials (Batch 1)

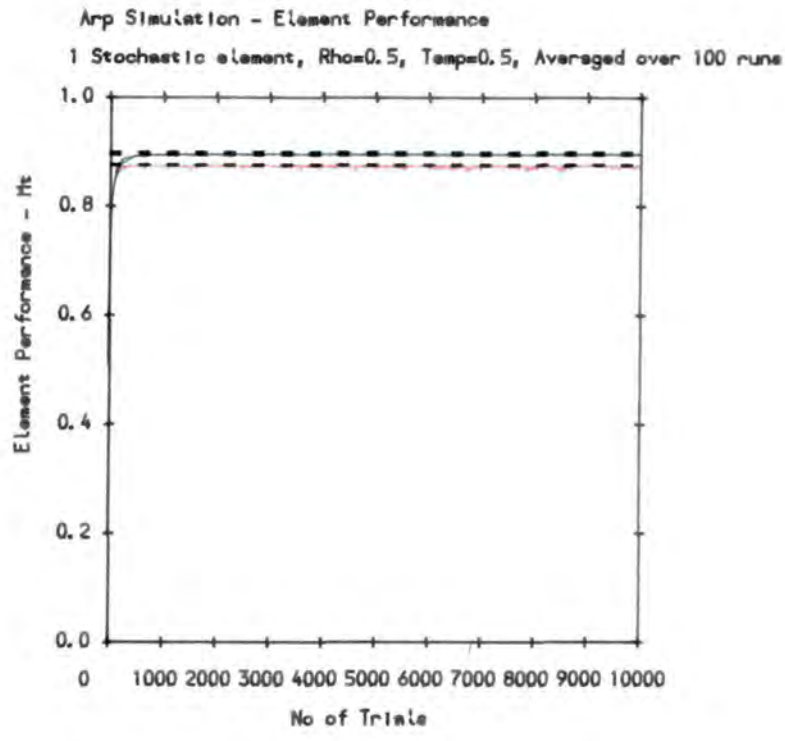


Figure 7.5a - M_t vs Number of Trials (Batch 10)

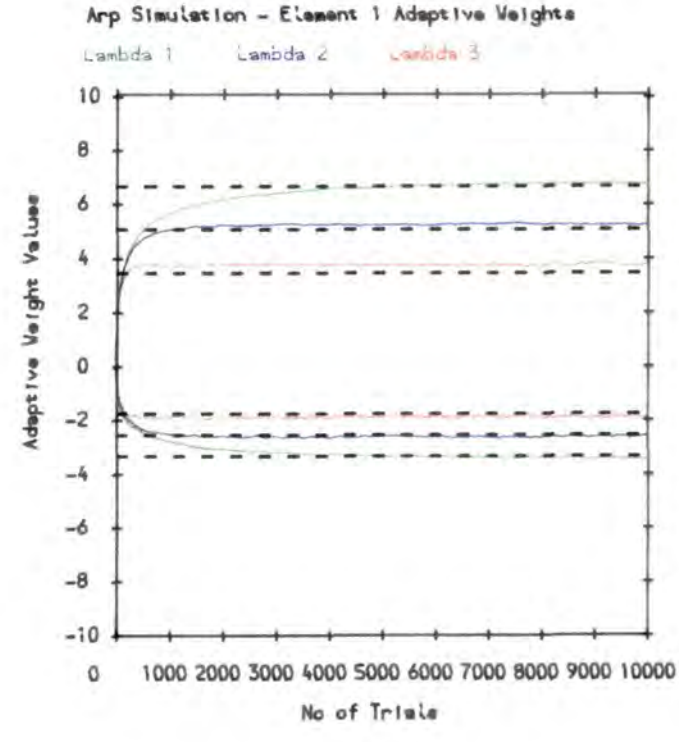


Figure 7.5b - w_1, w_2 vs Number of Trials (Batch 10)

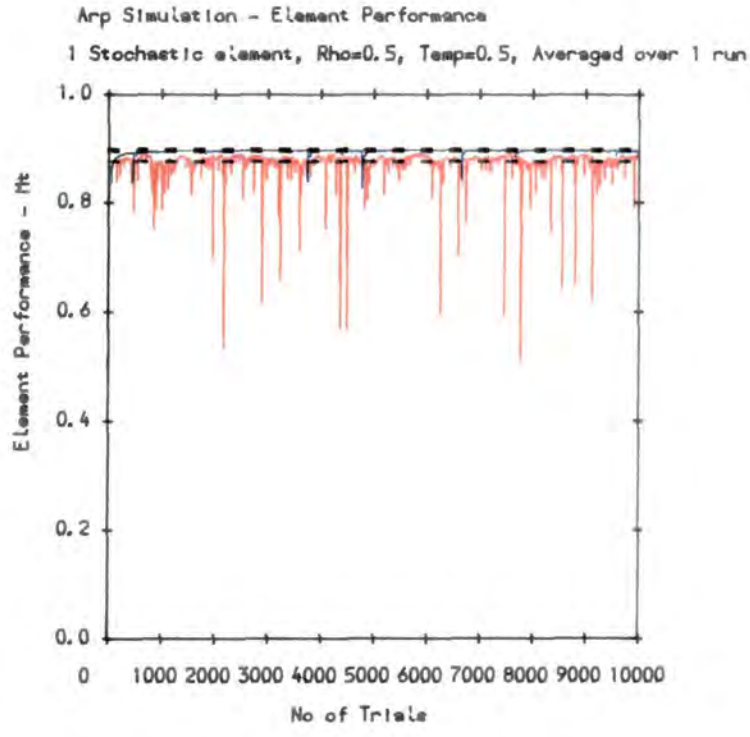


Figure 7.5c - M_t vs Number of Trials (Batch 10)

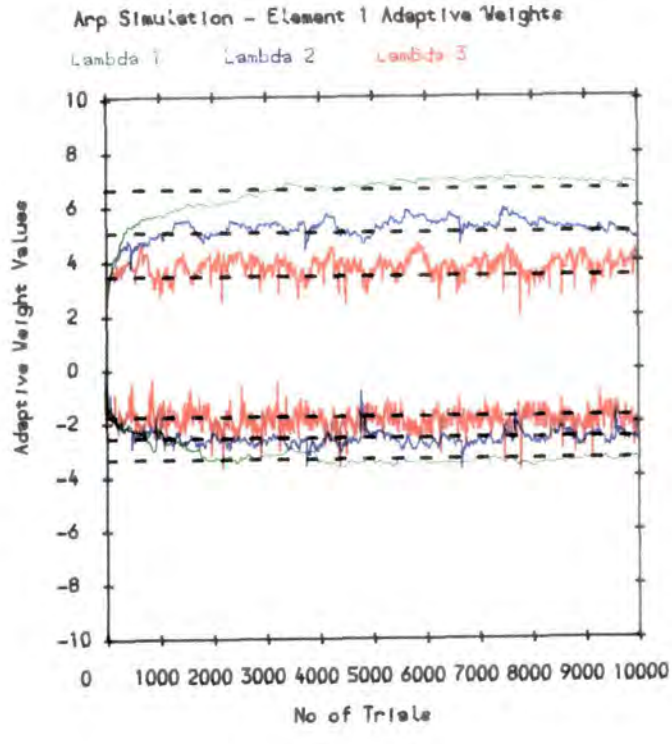


Figure 7.5d - w_1, w_2 vs Number of Trials (Batch 10)

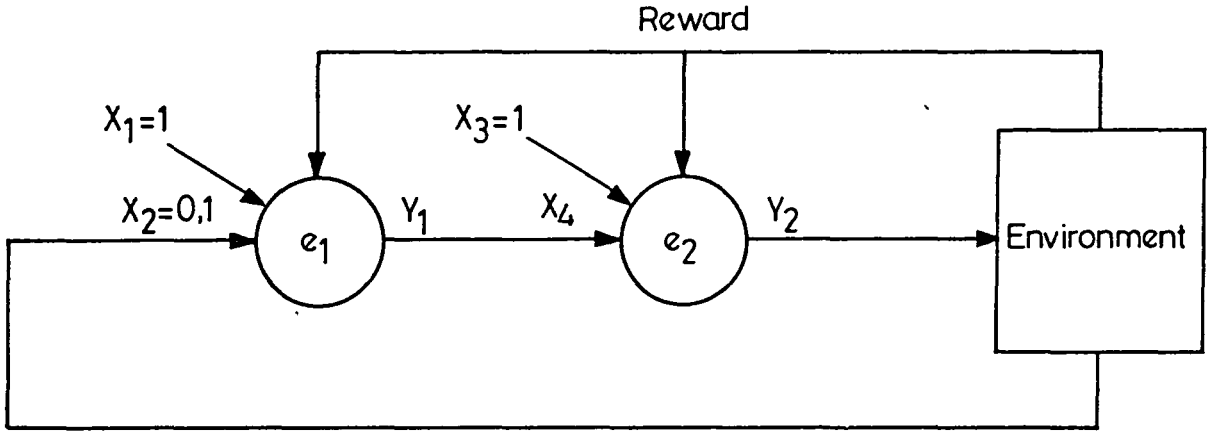


Figure 7.6 A_{R-P} Elements - linear configuration

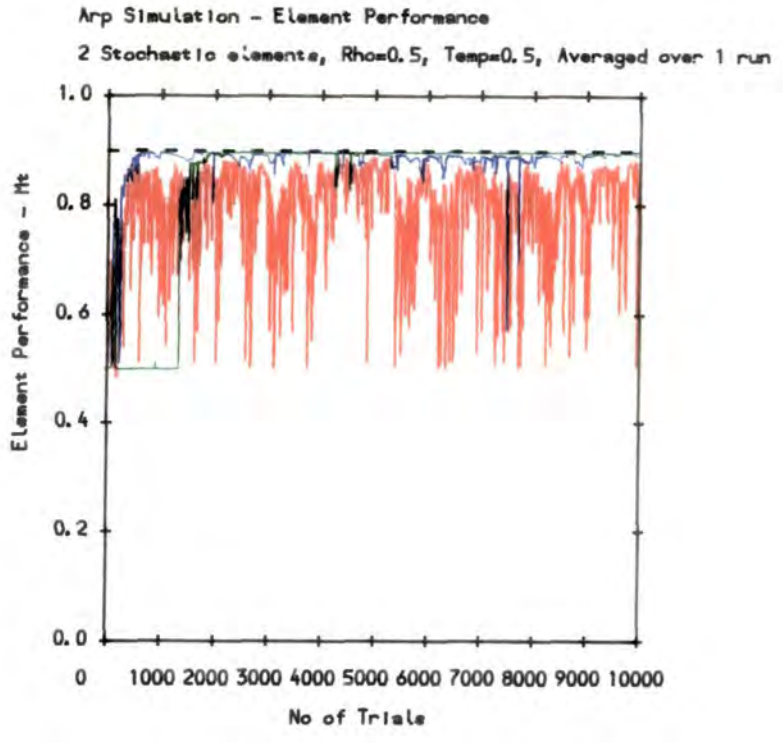


Figure 7.7a - M_t vs Number of Trials (Batch 1)

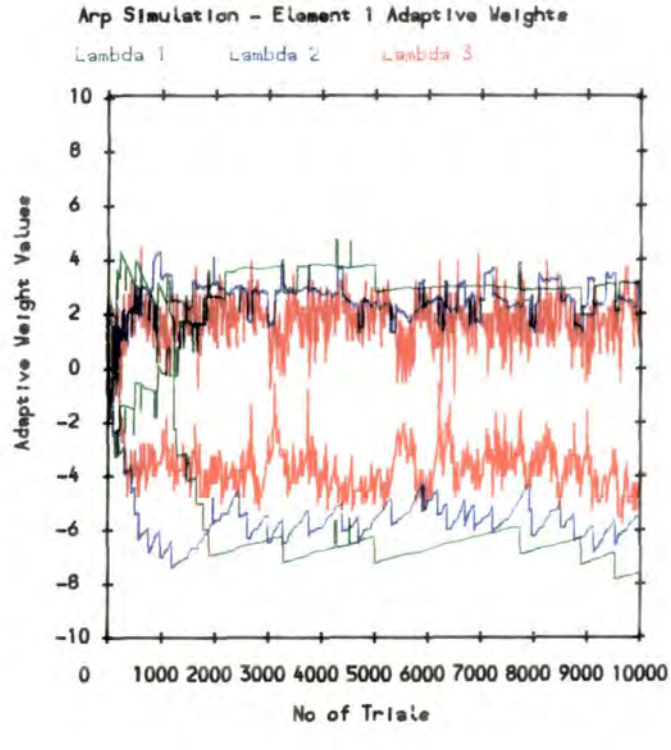


Figure 7.7b - w_{11}, w_{12} vs Number of Trials (Batch 1)

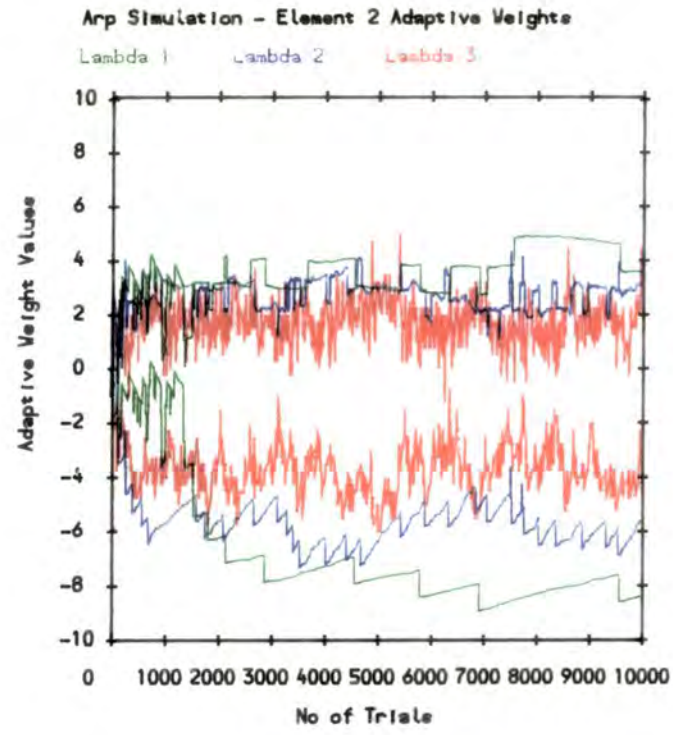


Figure 7.7c - w_{21}, w_{22} vs Number of Trials (Batch 1)

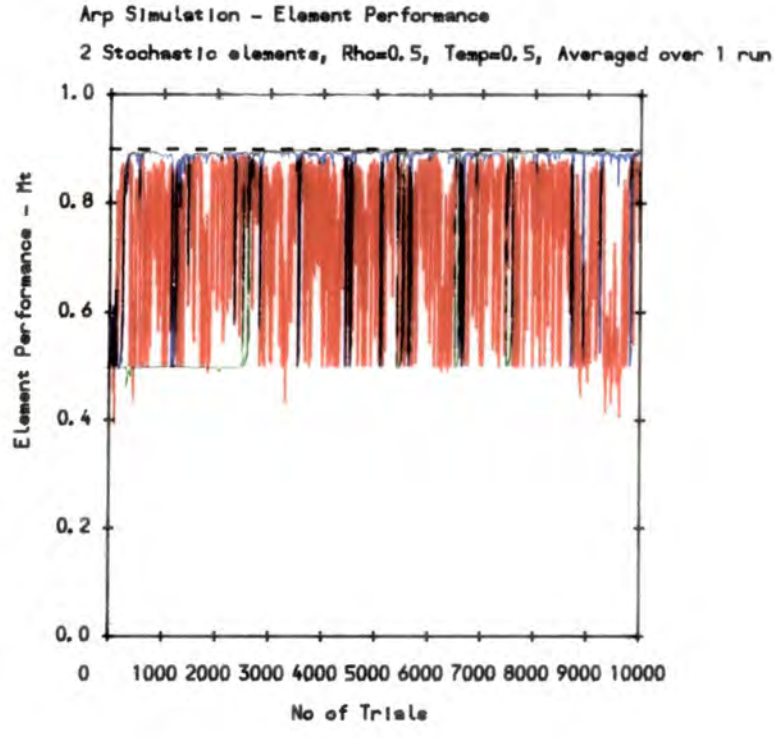


Figure 7.8a - M_t vs Number of Trials (Batch 10)

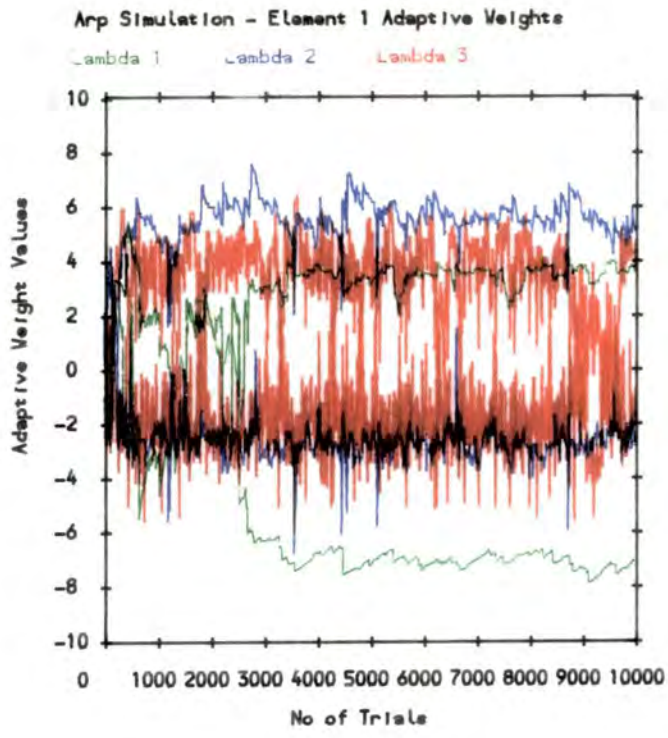


Figure 7.8b - w_{11}, w_{12} vs Number of Trials (Batch 10)

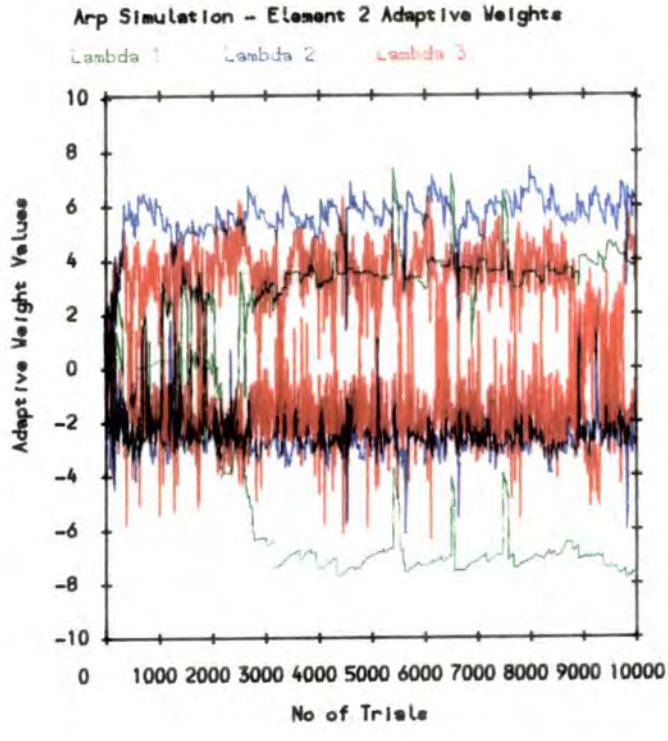


Figure 7.8c - w_{21}, w_{22} vs Number of Trials (Batch 10)

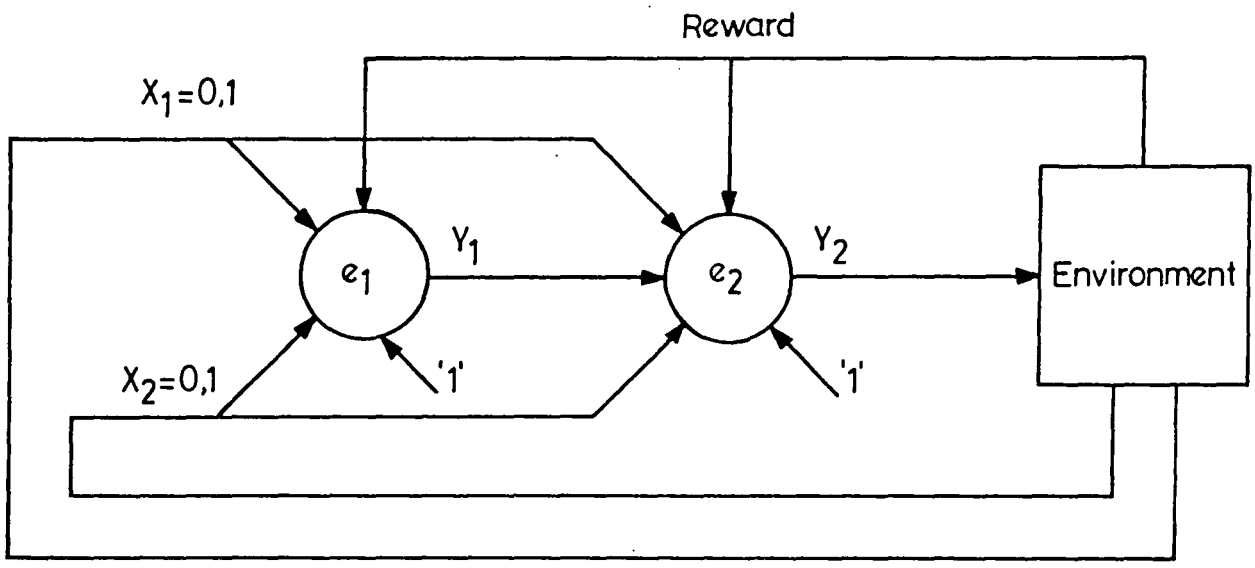


Figure 7.9 A_{R-P} Elements - parallel configuration

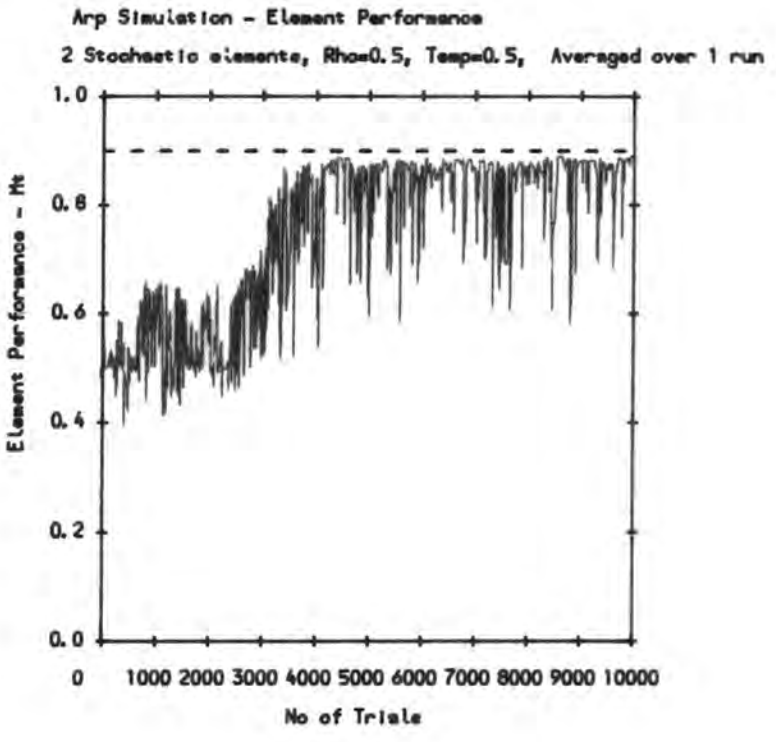


Figure 7.10a - M_t vs Number of Trials (Batch 1)

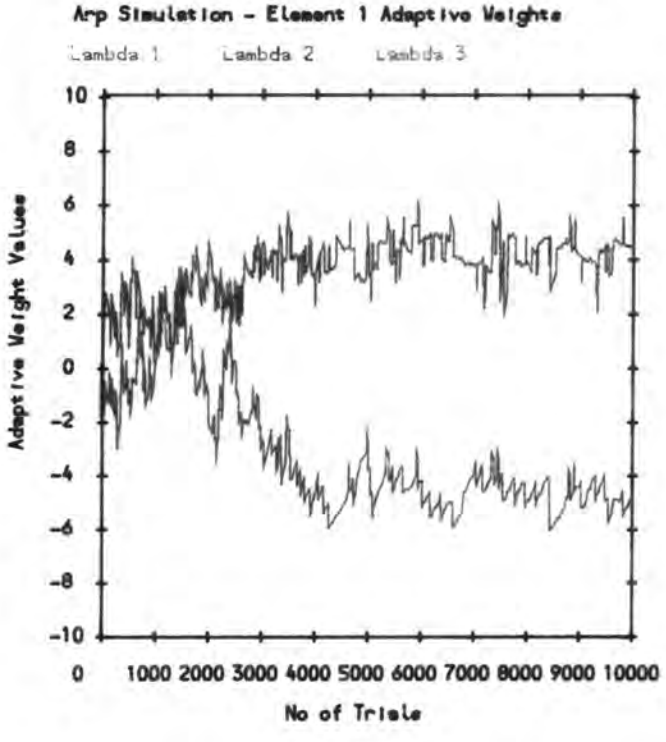


Figure 7.10b - w_{11}, w_{12} vs Number of Trials (Batch 1)

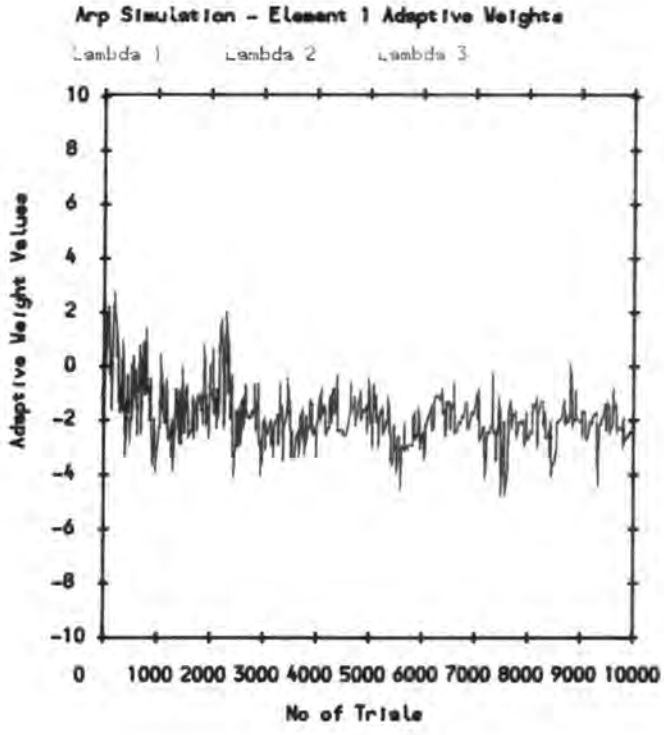


Figure 7.10c - w_{13} vs Number of Trials (Batch 1)

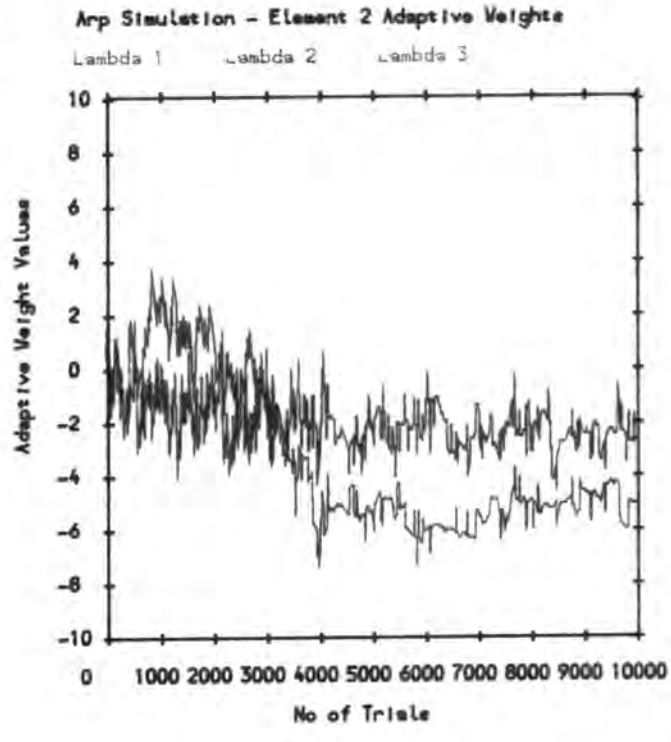


Figure 7.10d - w_{21}, w_{22} vs Number of Trials (Batch 1)

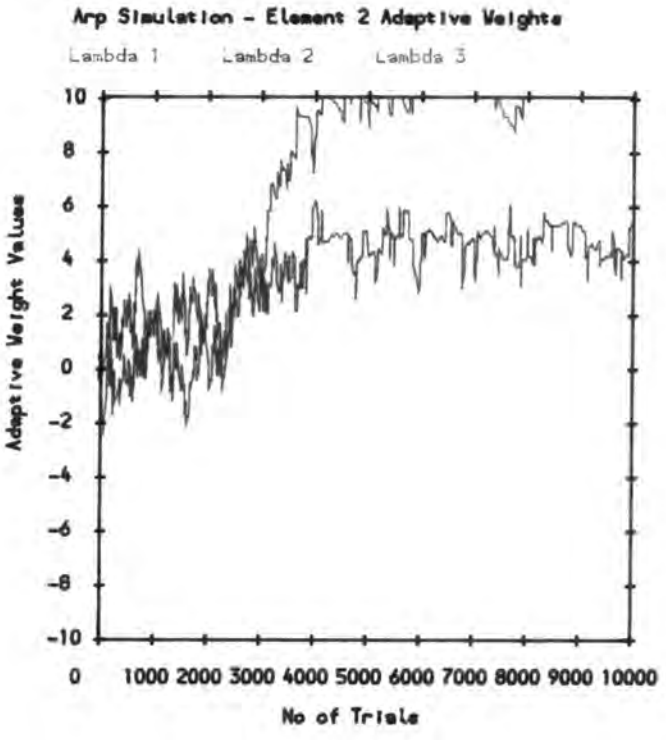


Figure 7.10e - w_{23}, w_{24} vs Number of Trials (Batch 1)

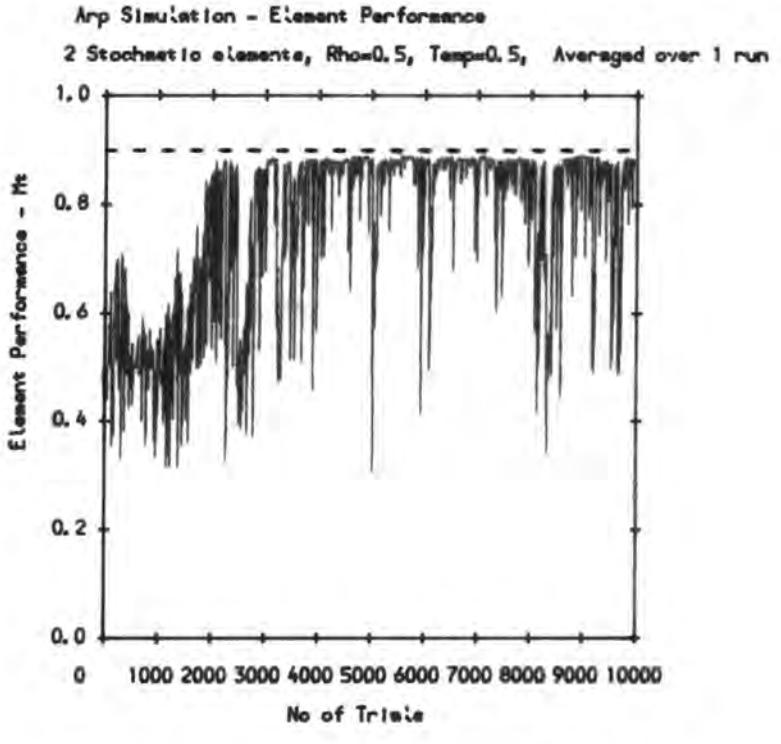


Figure 7.11a - M_t vs Number of Trials (Batch 10)

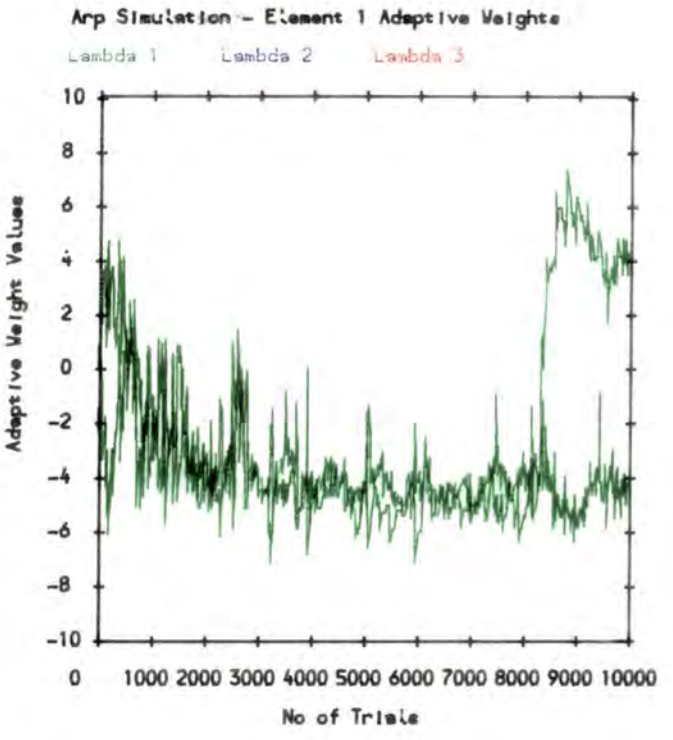


Figure 7.11b - w_{11}, w_{12} vs Number of Trials (Batch 10)

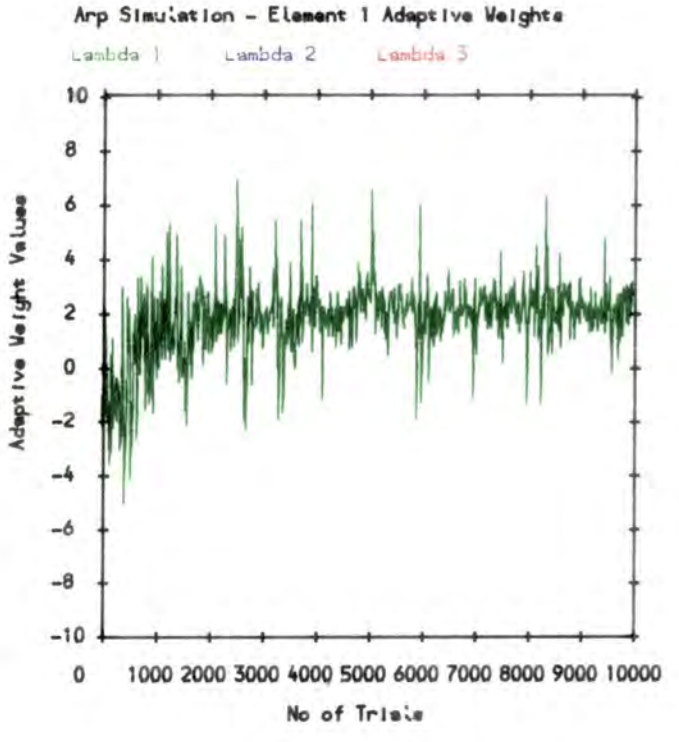


Figure 7.11c - w_{13} vs Number of Trials (Batch 10)

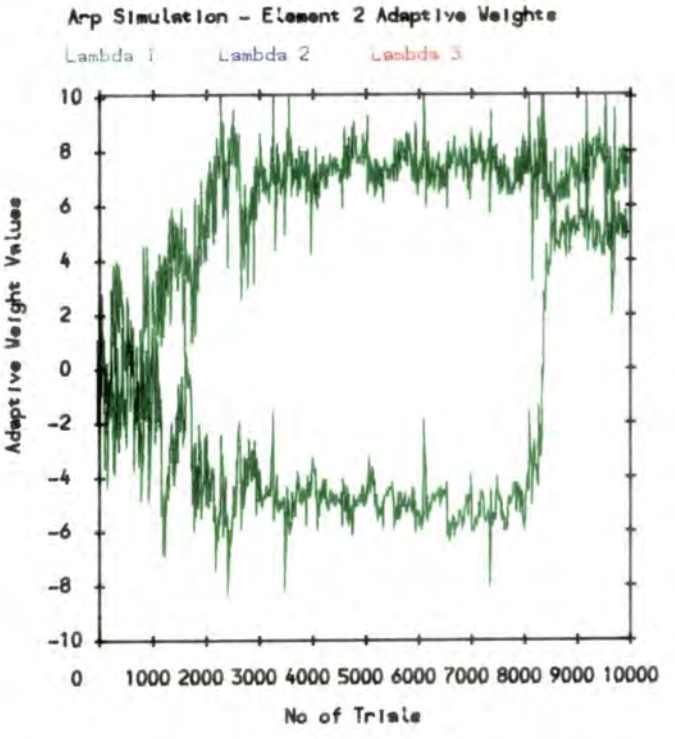


Figure 7.11d - w_{21}, w_{22} vs Number of Trials (Batch 10)

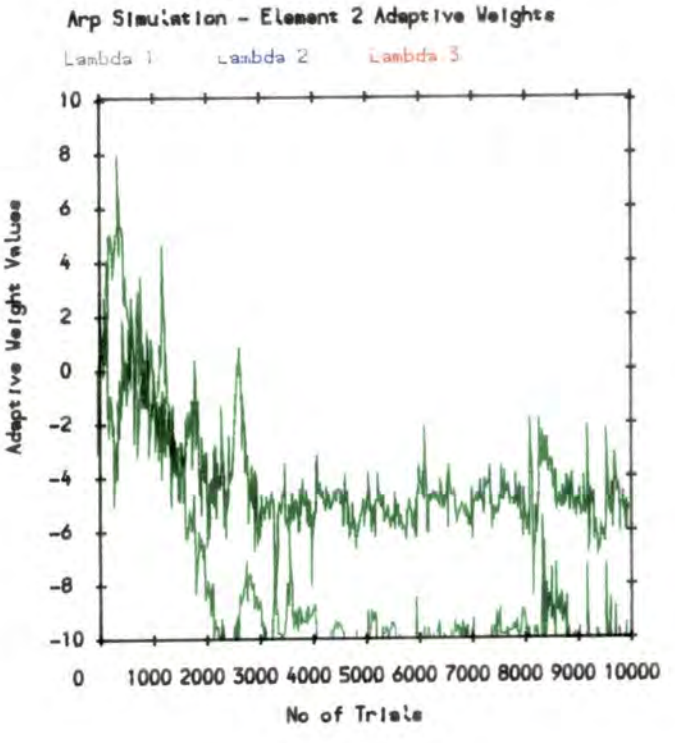


Figure 7.11e - w_{23}, w_{24} vs Number of Trials (Batch 10)

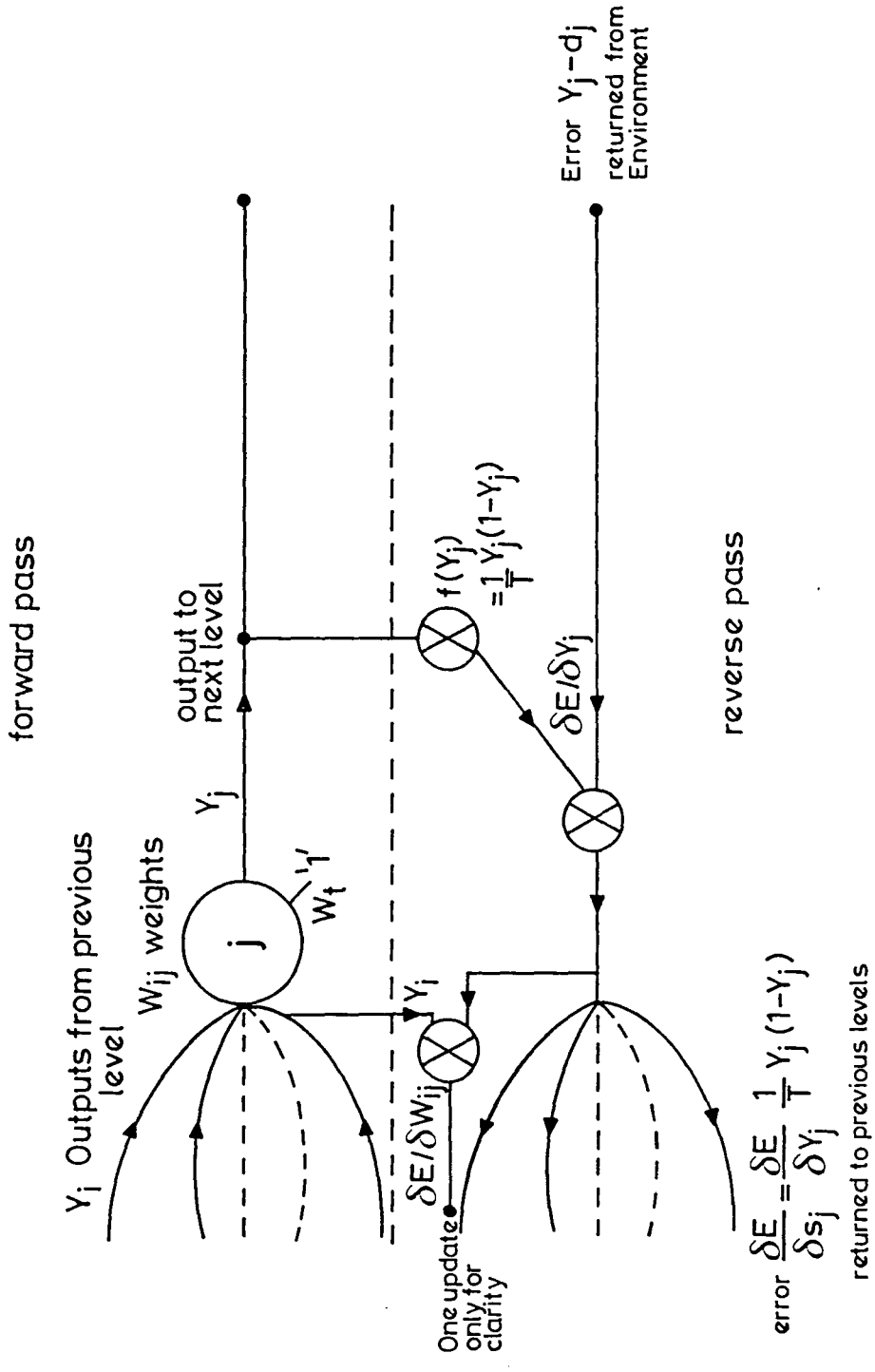


Figure 7.12a Deterministic Error backpropagation – Final Layer Element

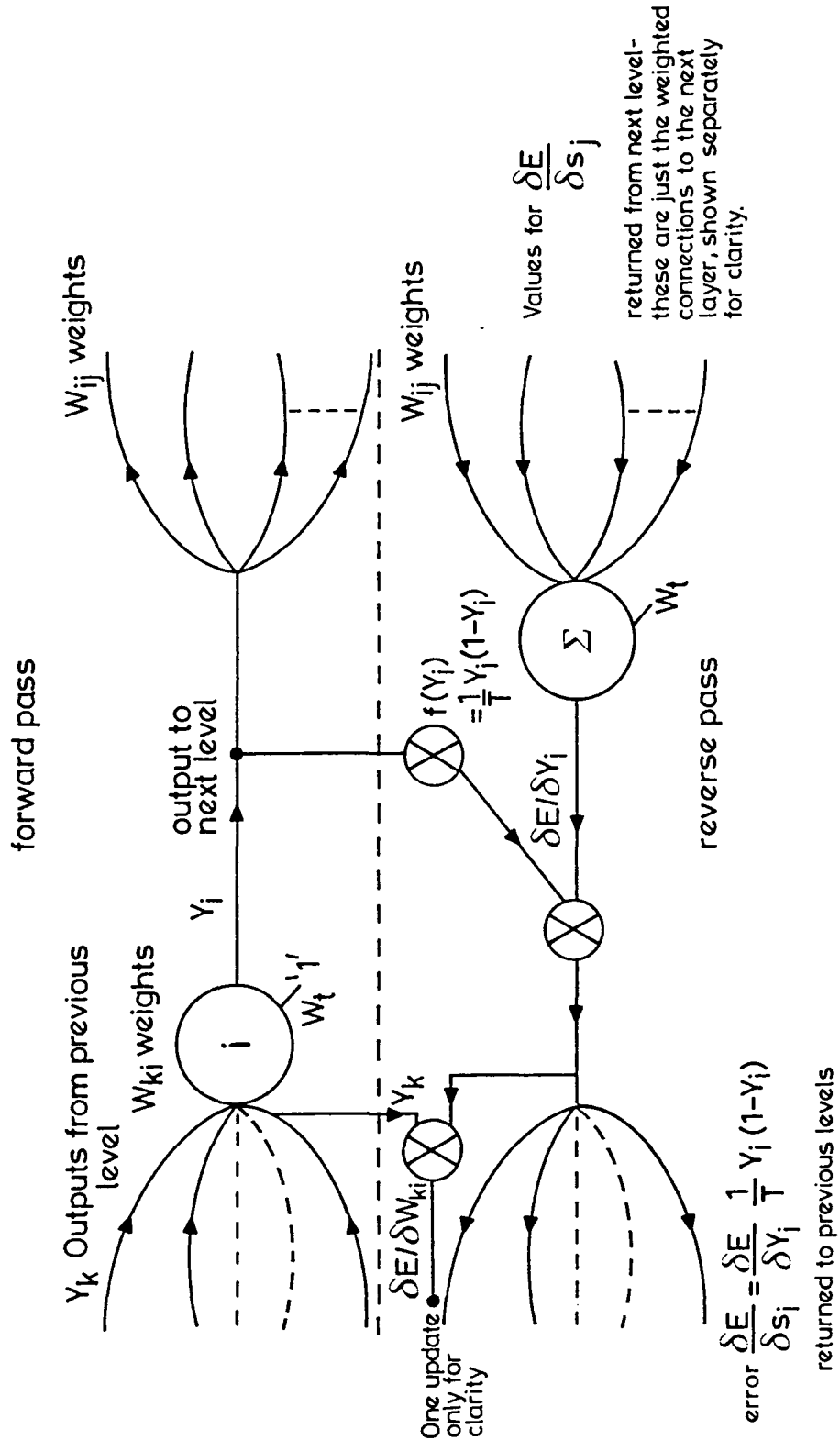


Figure 7.12b Deterministic Error backpropagation – Previous Layer Element

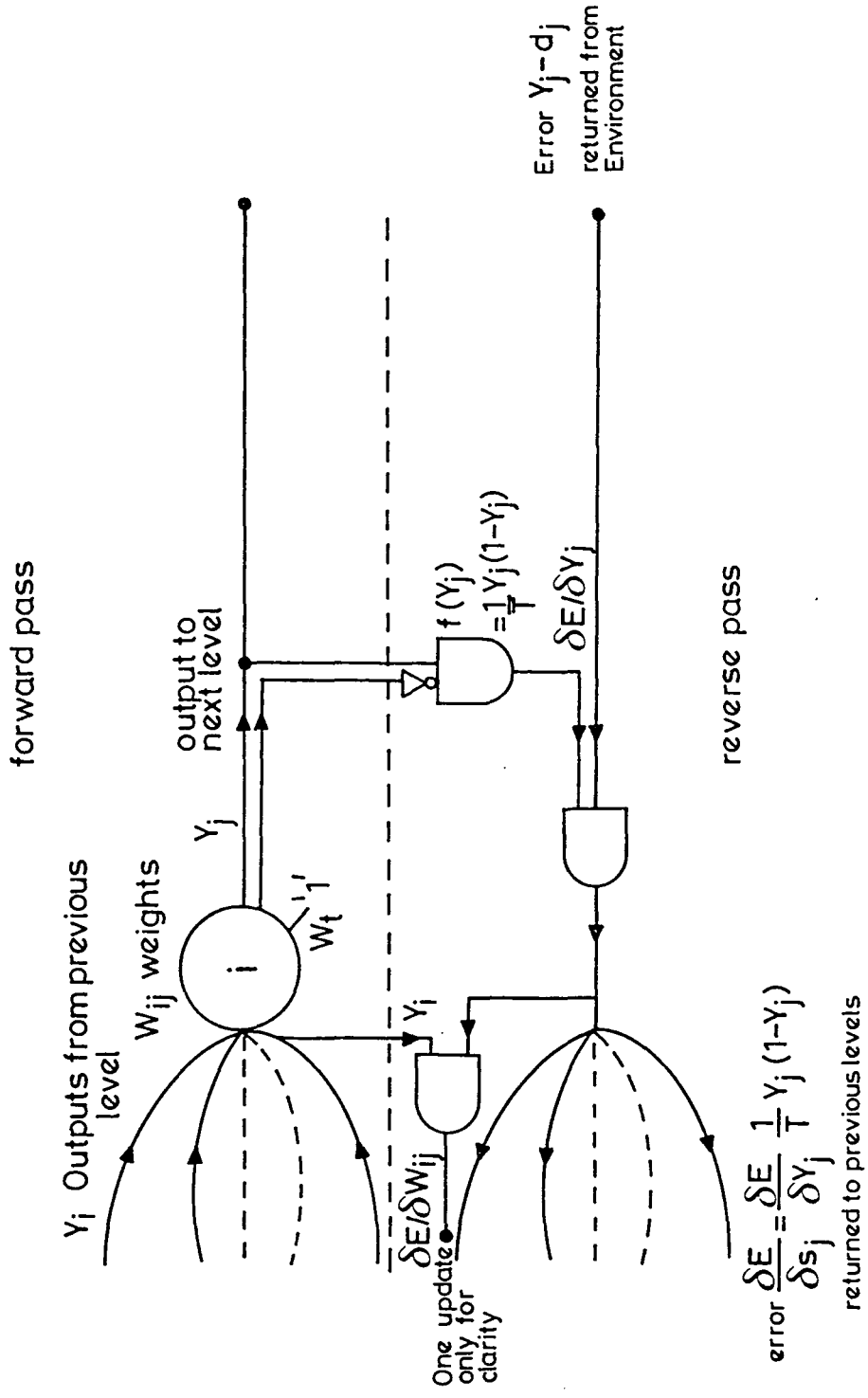


Figure 7.13a Stochastic Error backpropagation – Final Layer Element

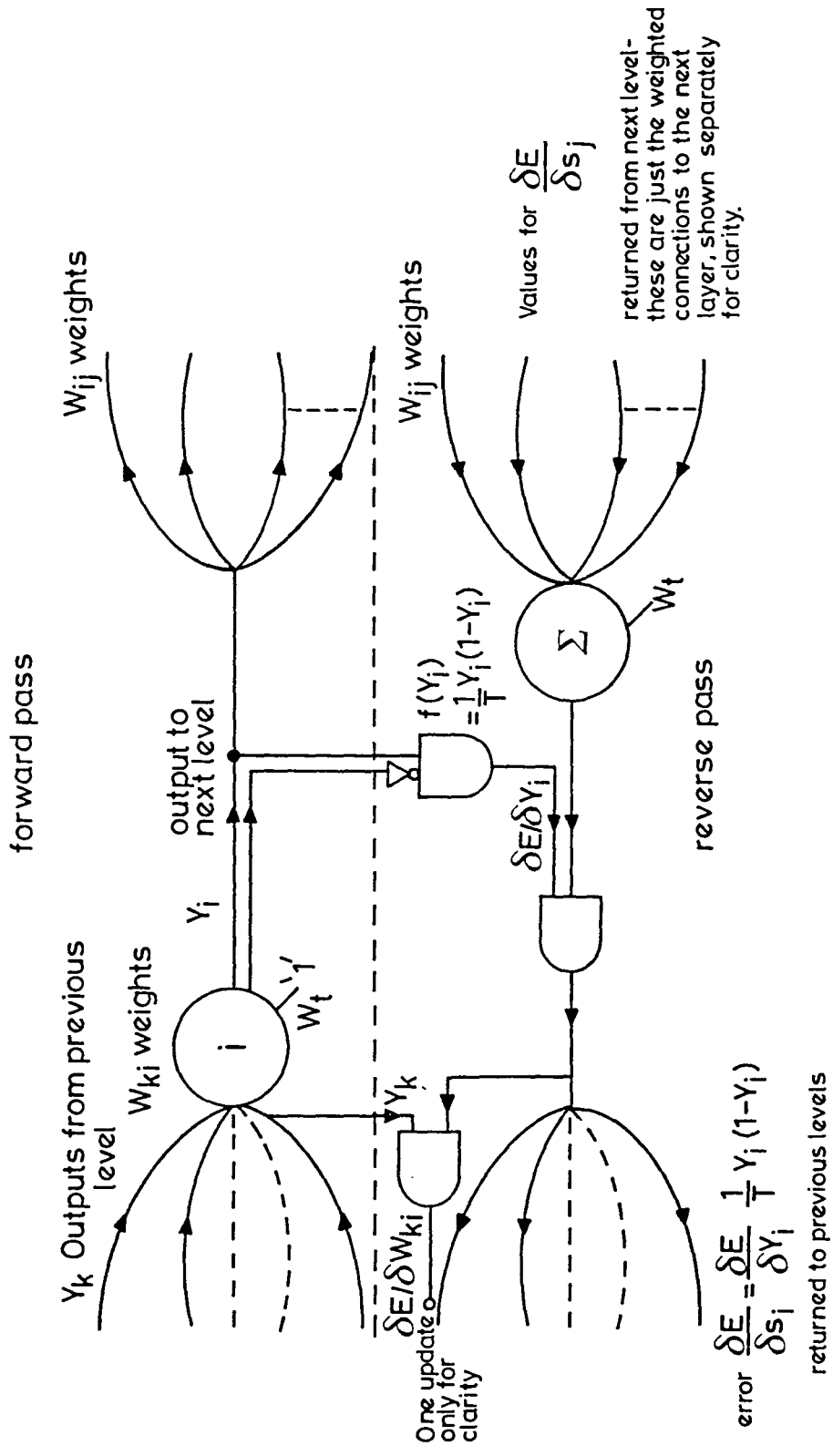


Figure 7.13b Stochastic Error backpropagation - Previous Layer Element

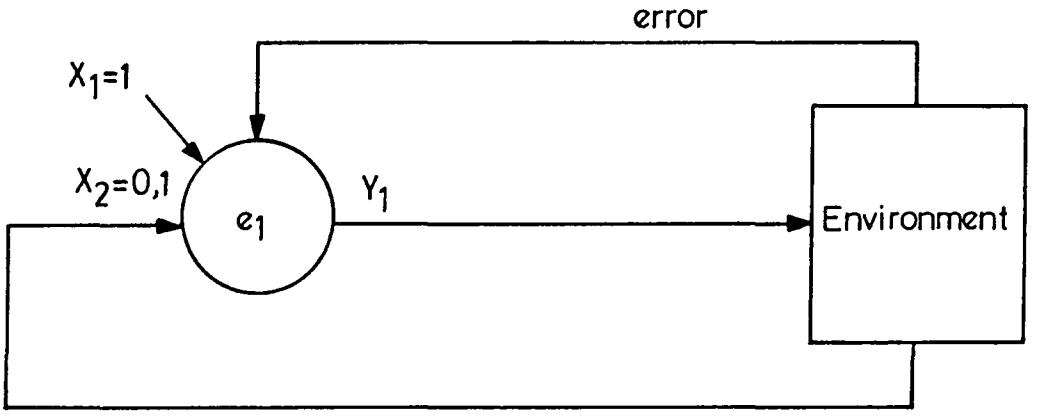


Figure 7.14 Error backpropagation element, Input and Environment

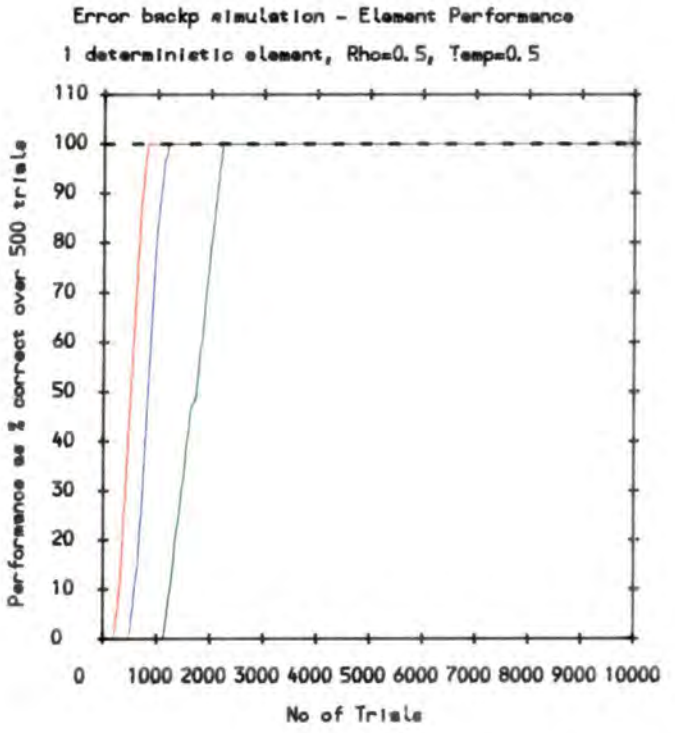


Figure 7.15a - % performance vs Number of Trials

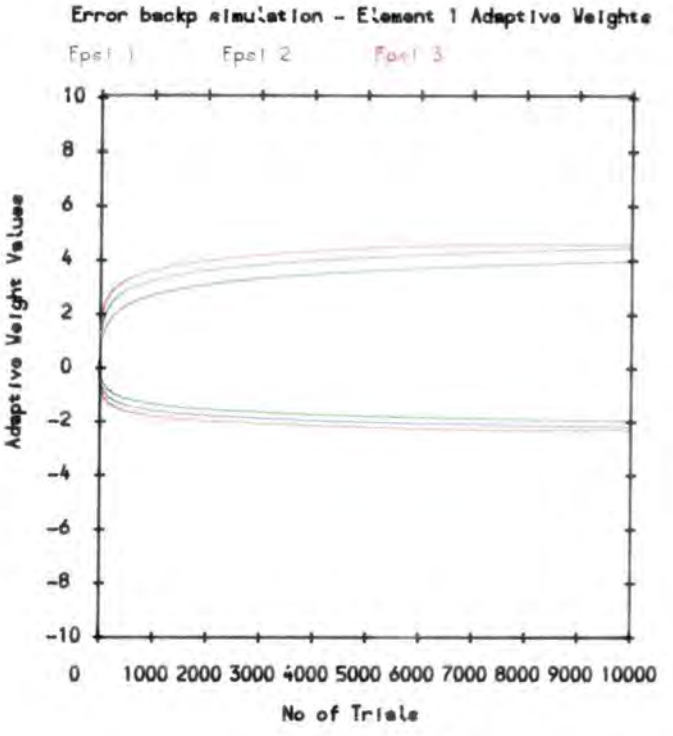


Figure 7.15b - w_{11}, w_{12} vs Number of Trials

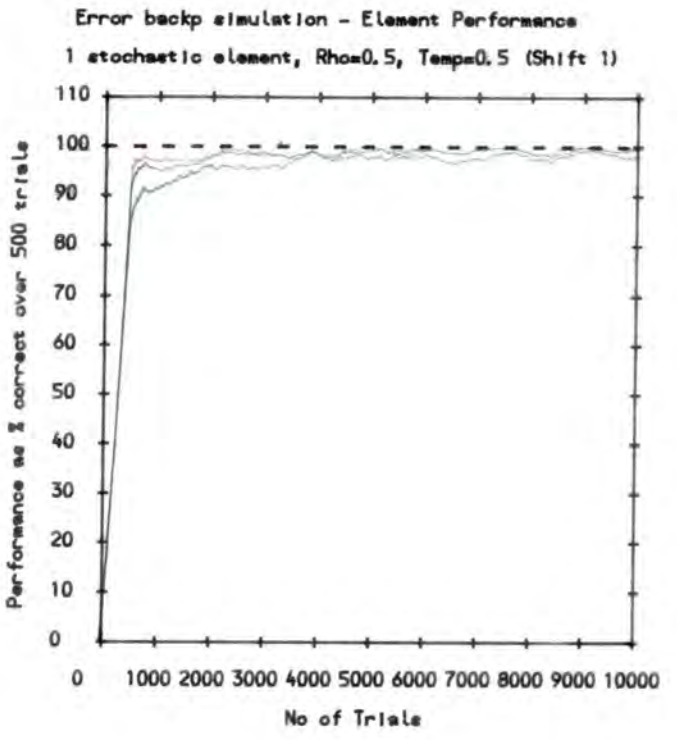


Figure 7.16a - % performance vs Number of Trials (Shift 1)

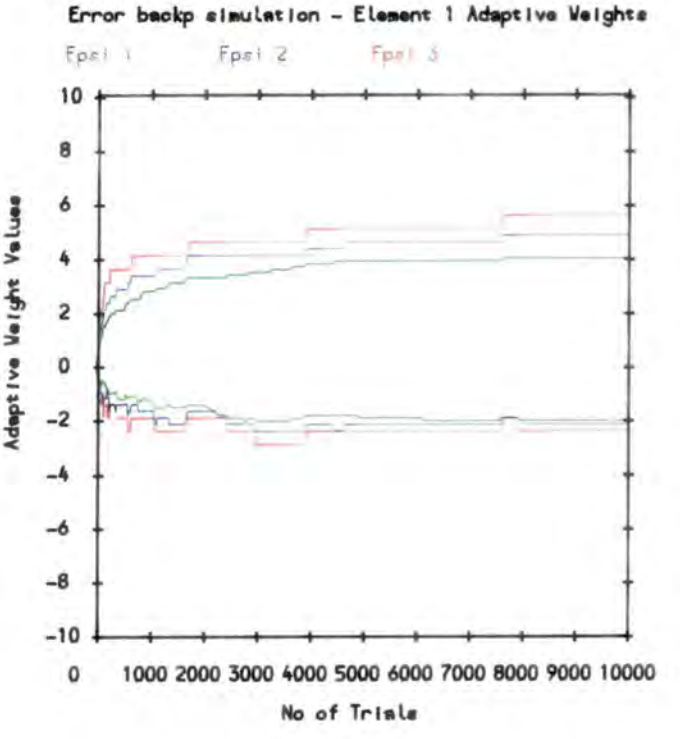


Figure 7.16b - w_{11}, w_{12} vs Number of Trials (Shift 1)

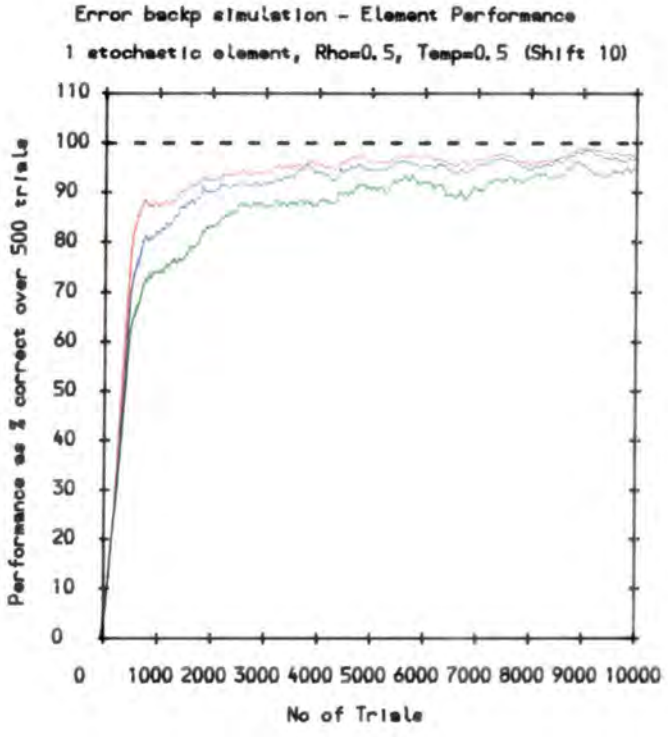


Figure 7.17a - % performance vs Number of Trials (Shift 10)

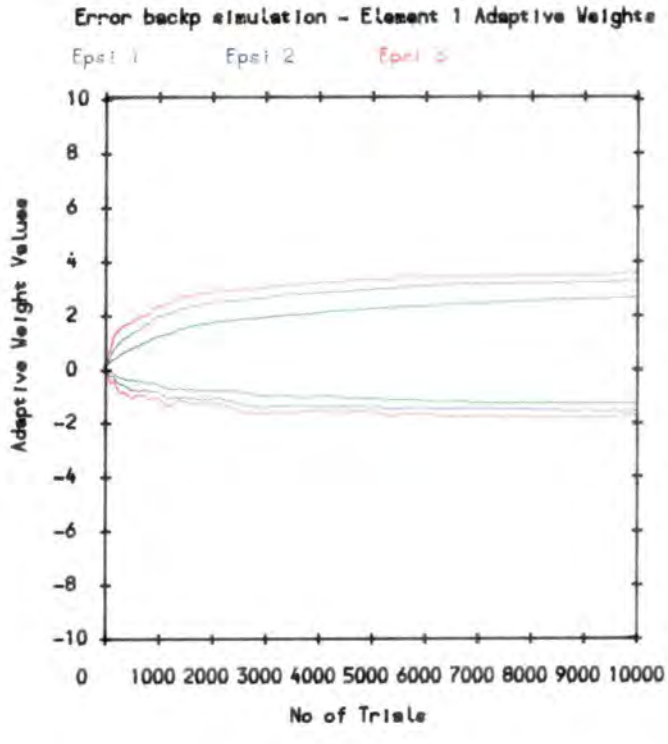


Figure 7.17b - w_{11}, w_{12} vs Number of Trials (Shift 10)

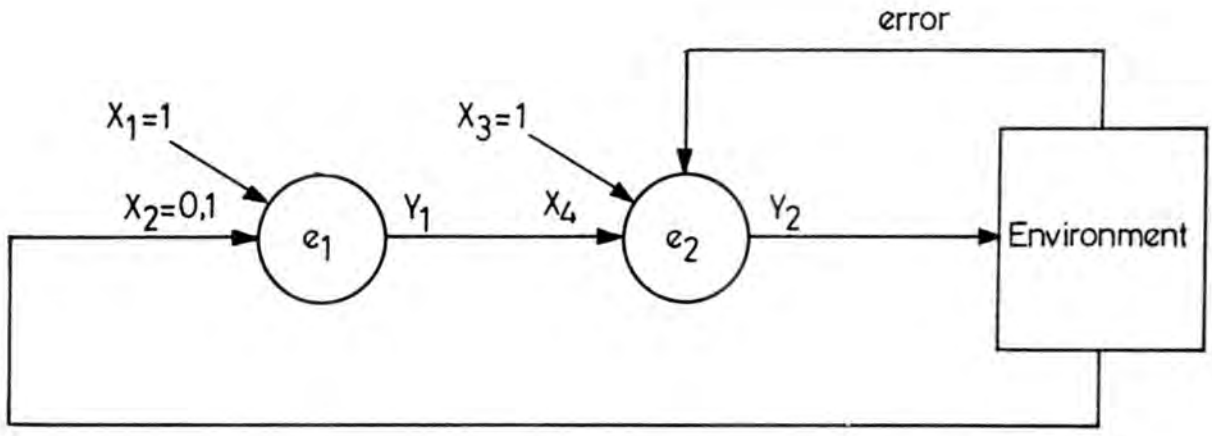


Figure 7.18 Error backpropagation elements, Input and Environment

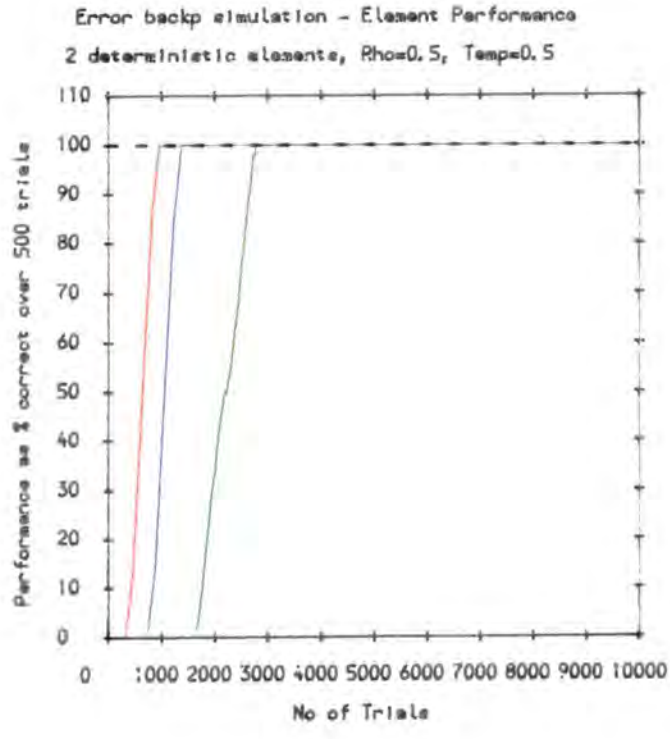


Figure 7.19a - % performance vs Number of Trials

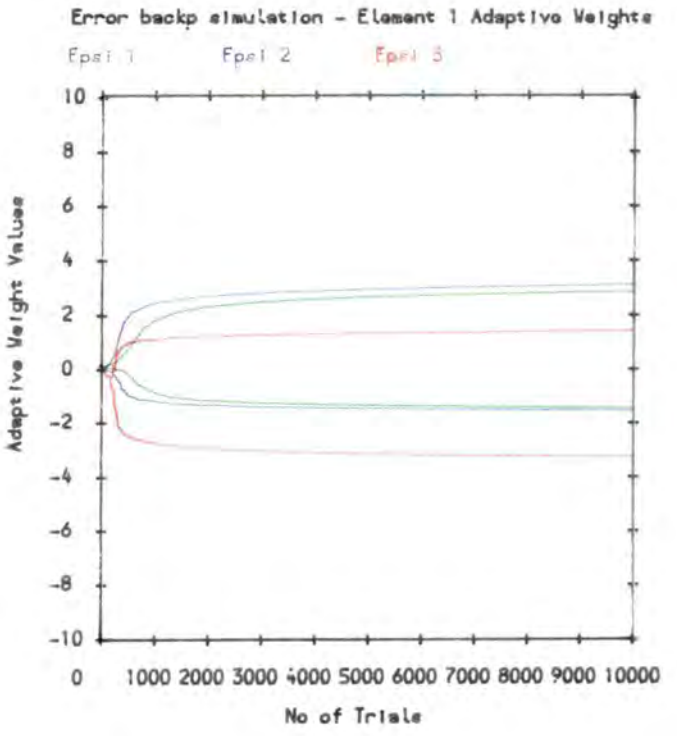


Figure 7.19b - w_{11}, w_{12} vs Number of Trials

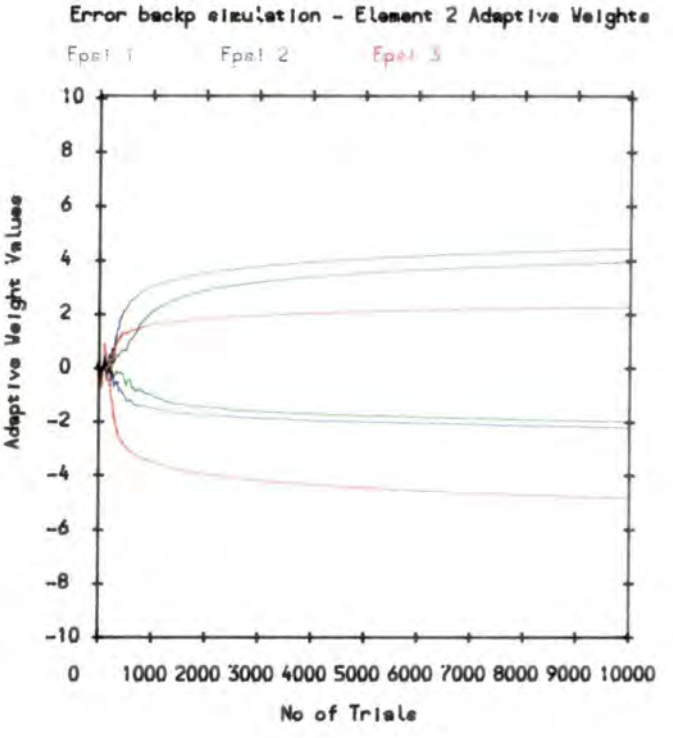


Figure 7.19c - w_{21}, w_{22} vs Number of Trials

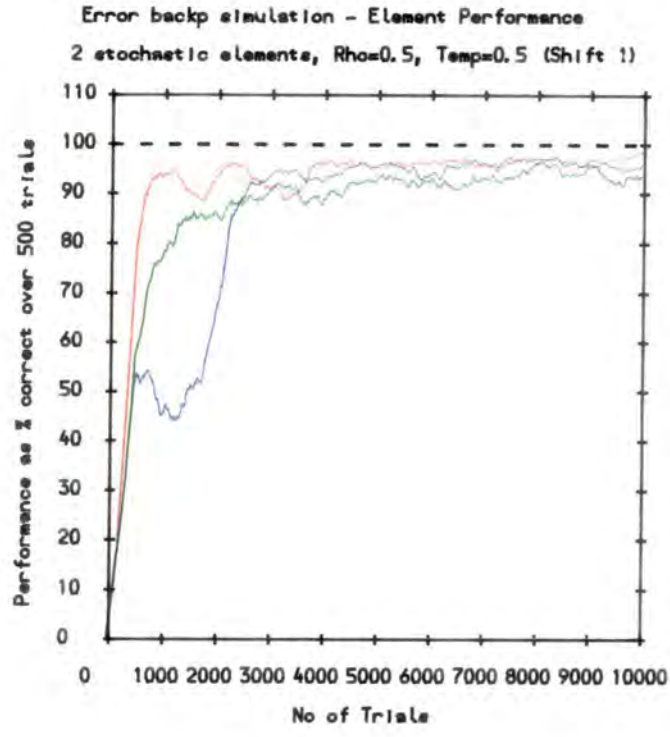


Figure 7.20a - % performance vs Number of Trials (Shift 1)

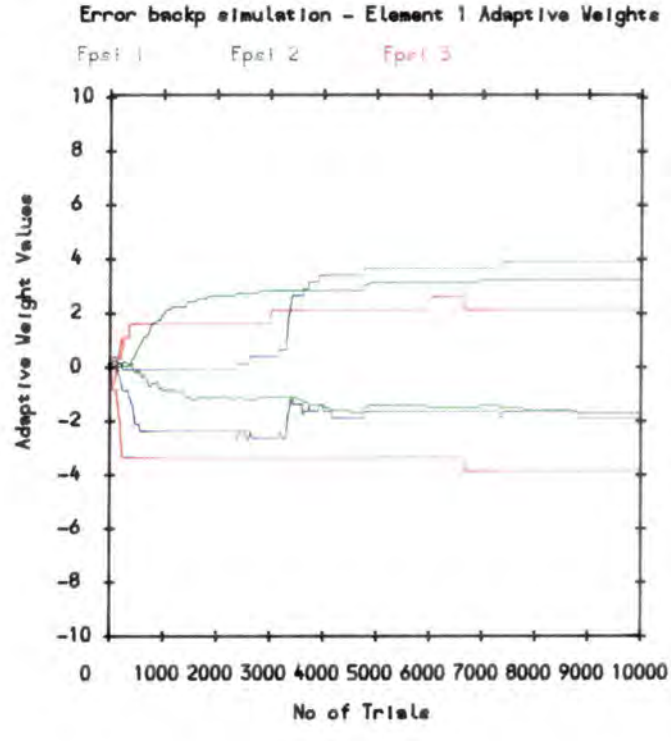


Figure 7.20b - w_{11}, w_{12} vs Number of Trials (Shift 1)

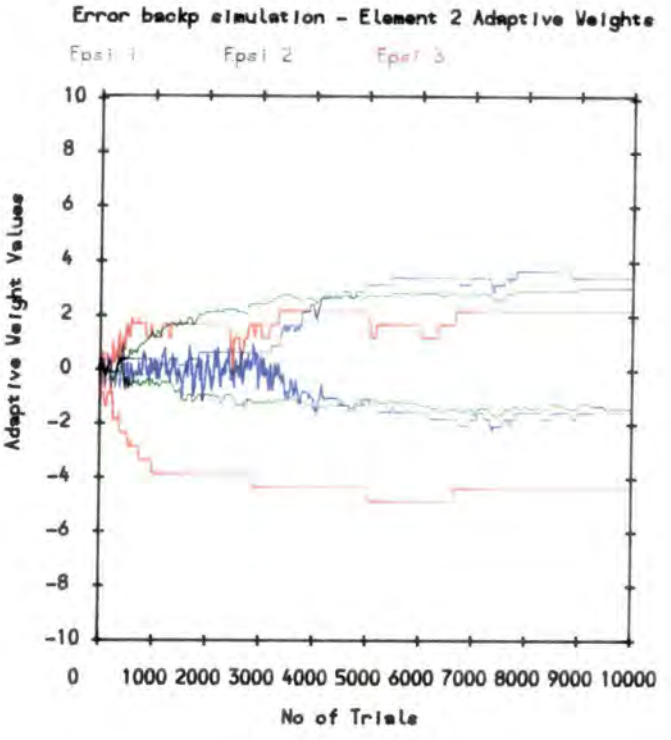


Figure 7.20c - w_{21}, w_{22} vs Number of Trials (Shift 1)

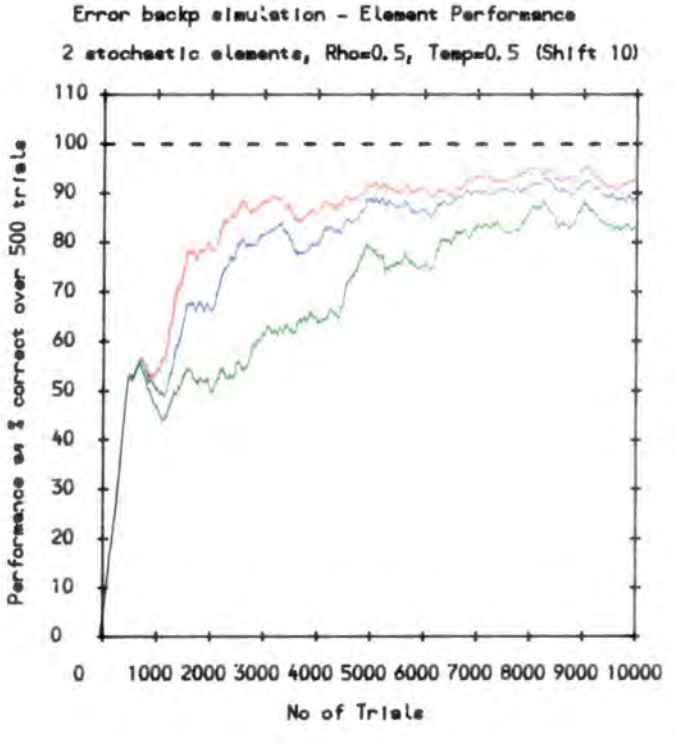


Figure 7.21a - % performance vs Number of Trials (Shift 10)

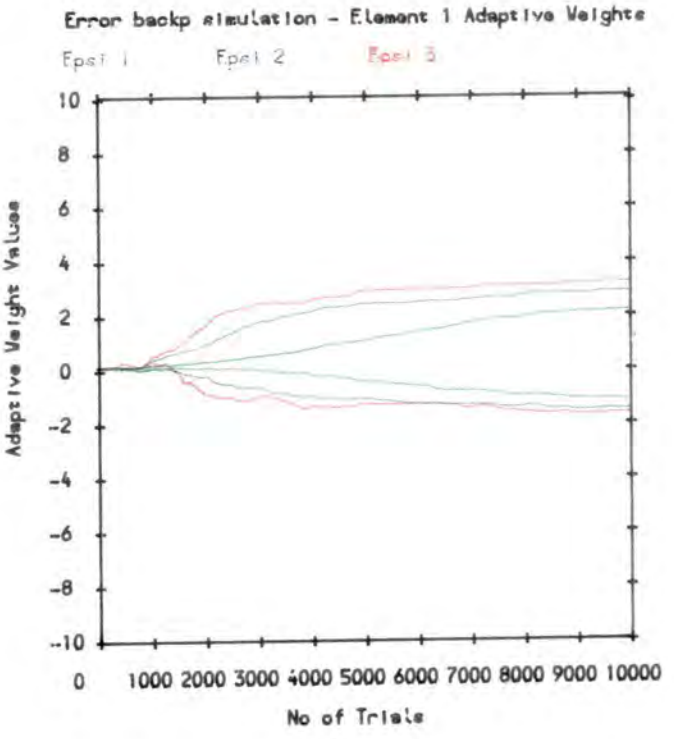


Figure 7.21b - w_{11}, w_{12} vs Number of Trials (Shift 10)

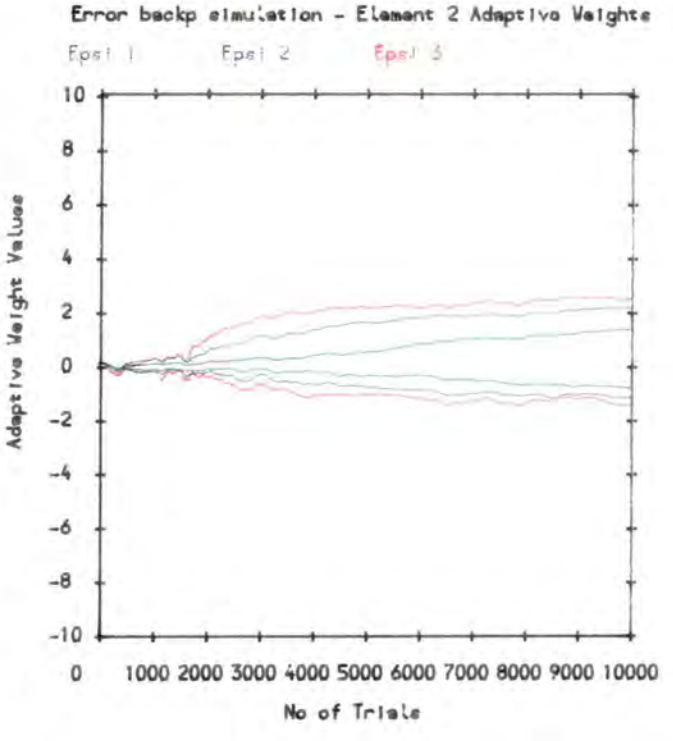


Figure 7.21c - w_{21}, w_{22} vs Number of Trials (Shift 10)

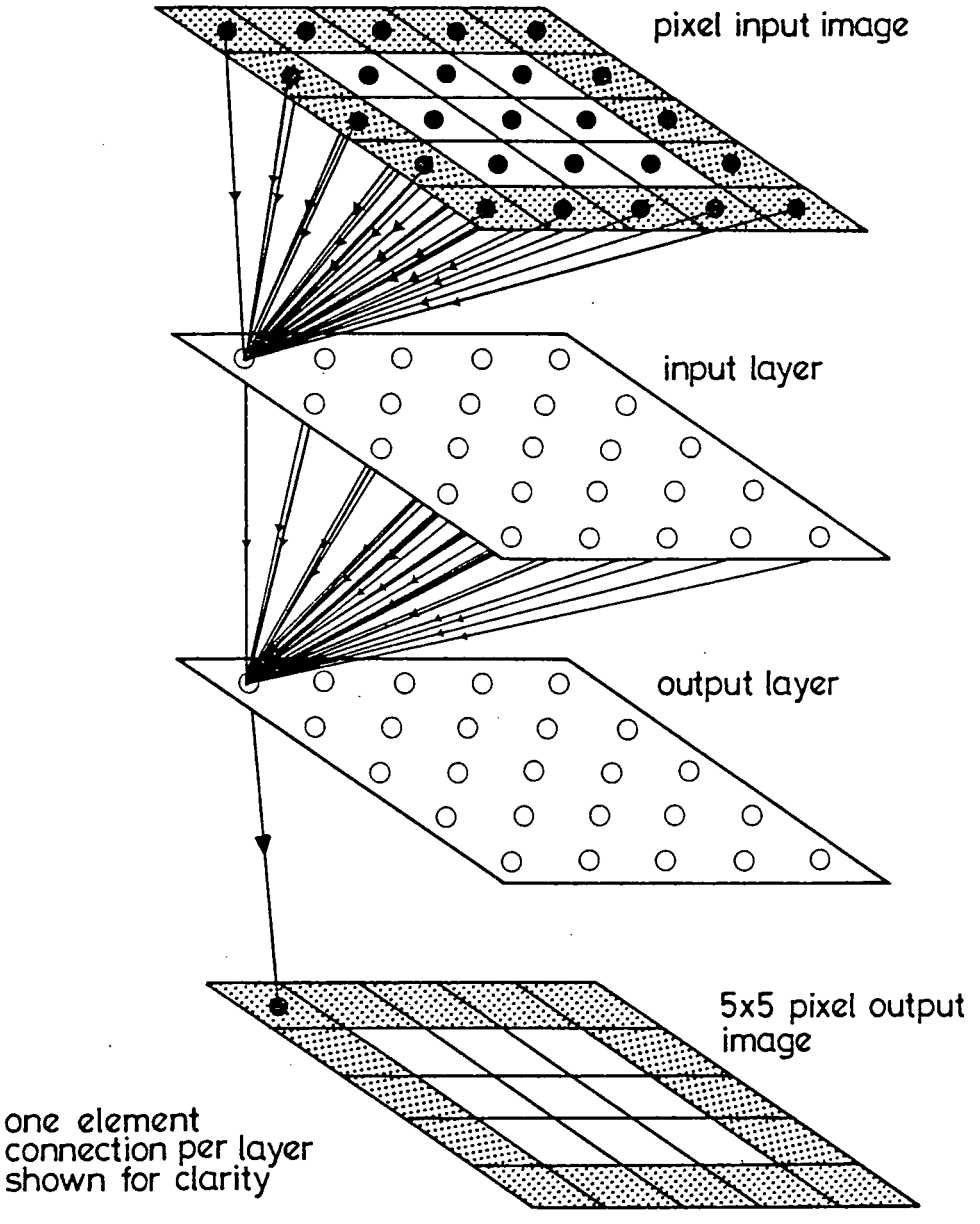


Figure 7.22 Two level 5 x 5 array zero image problem

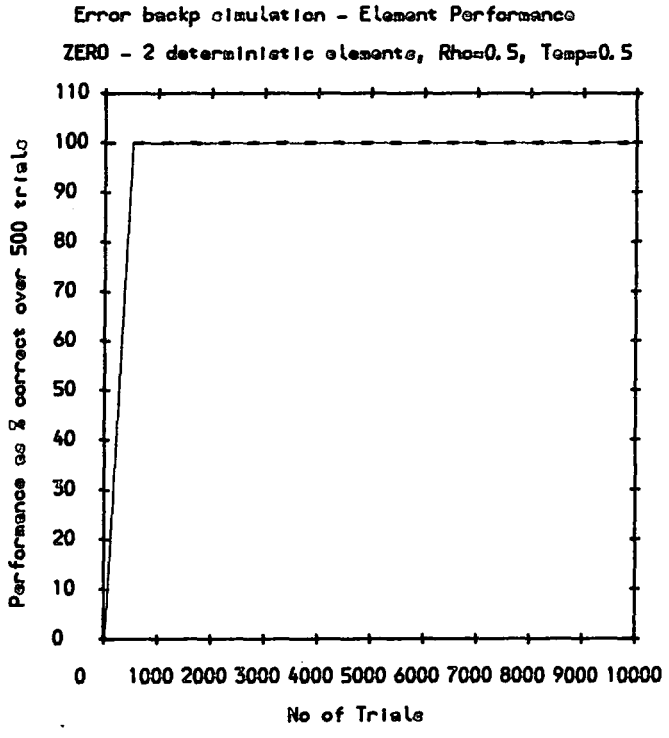


Figure 7.23a - % performance vs Number of Trials

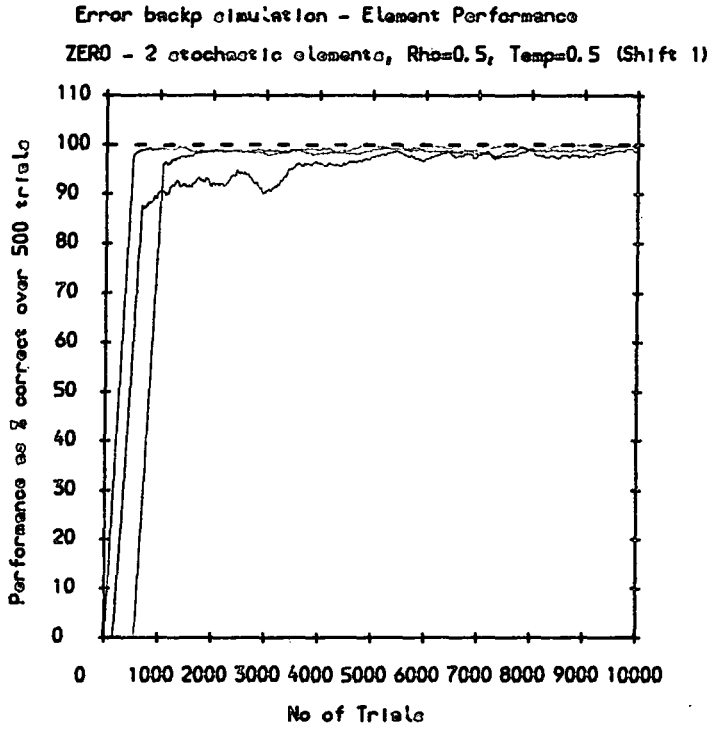


Figure 7.23b - % performance vs Number of Trials (Shift 1)

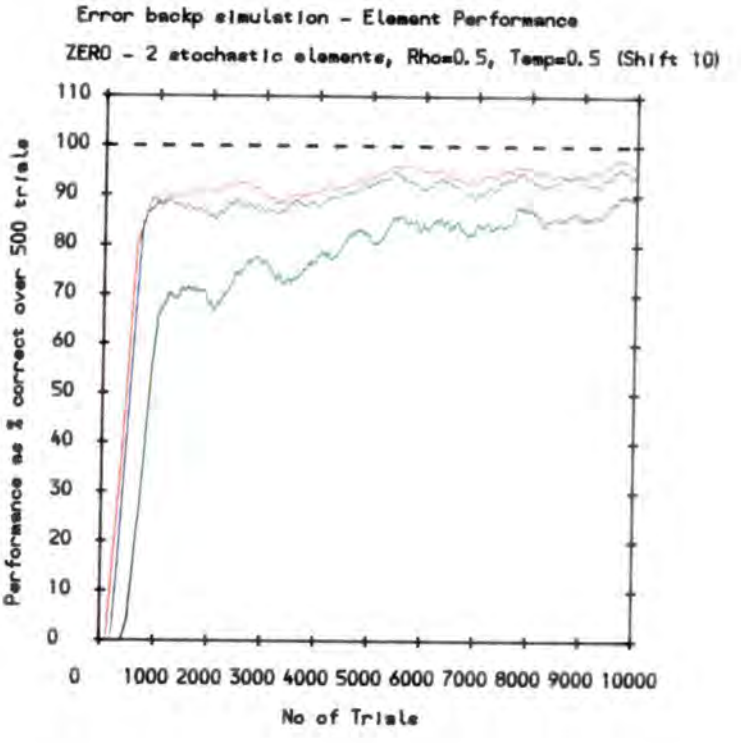


Figure 7.23c - % performance vs Number of Trials (Shift 10)

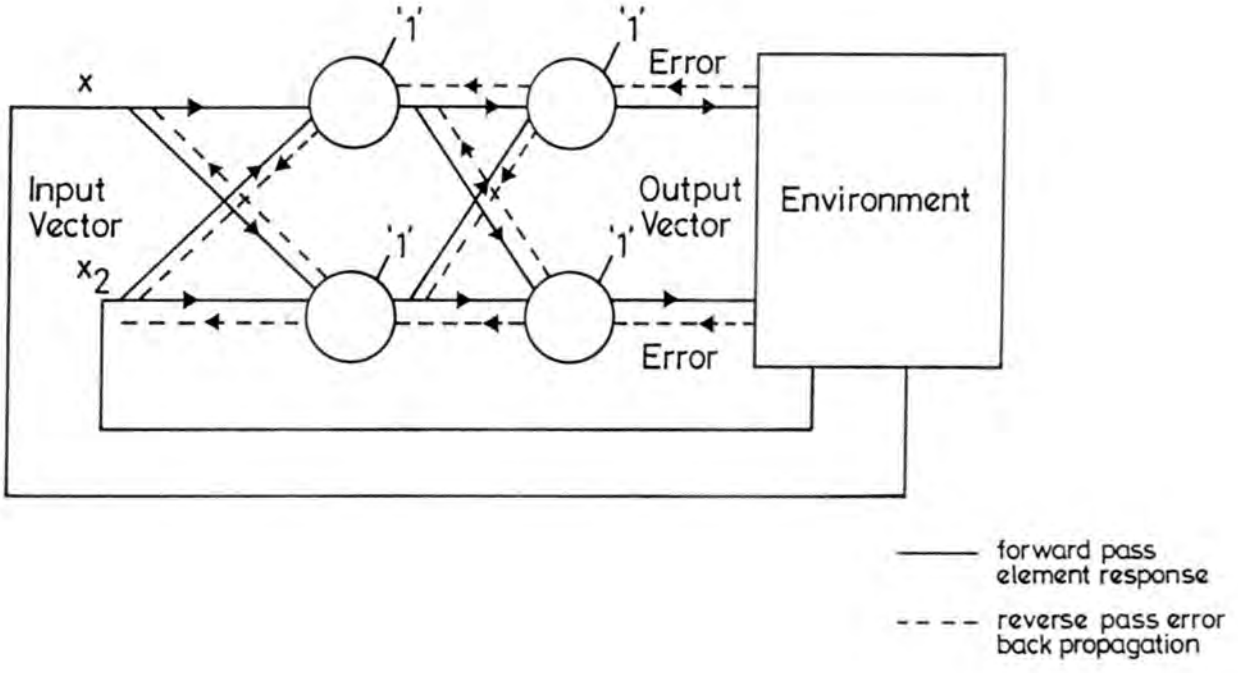


Figure 7.24 Error backpropagation array - Exclusive OR problem

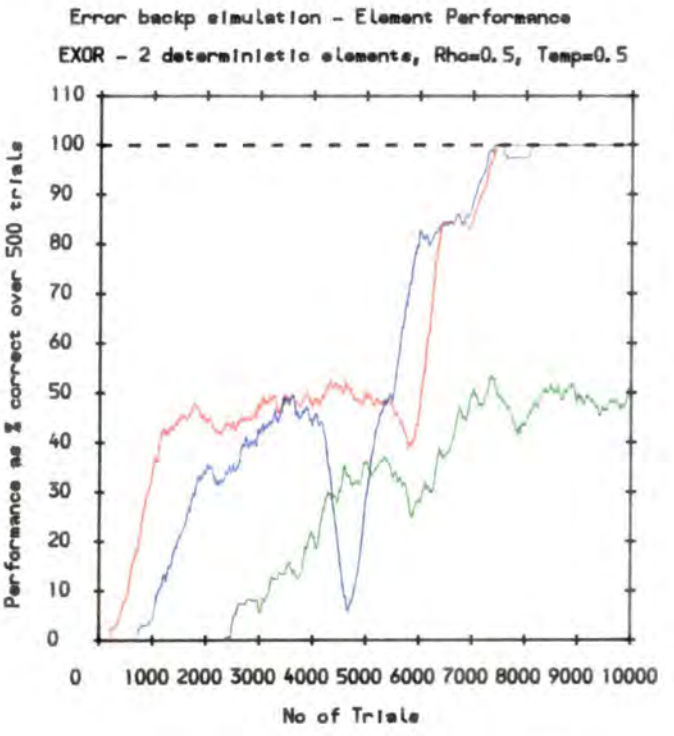


Figure 7.25a - % performance vs Number of Trials

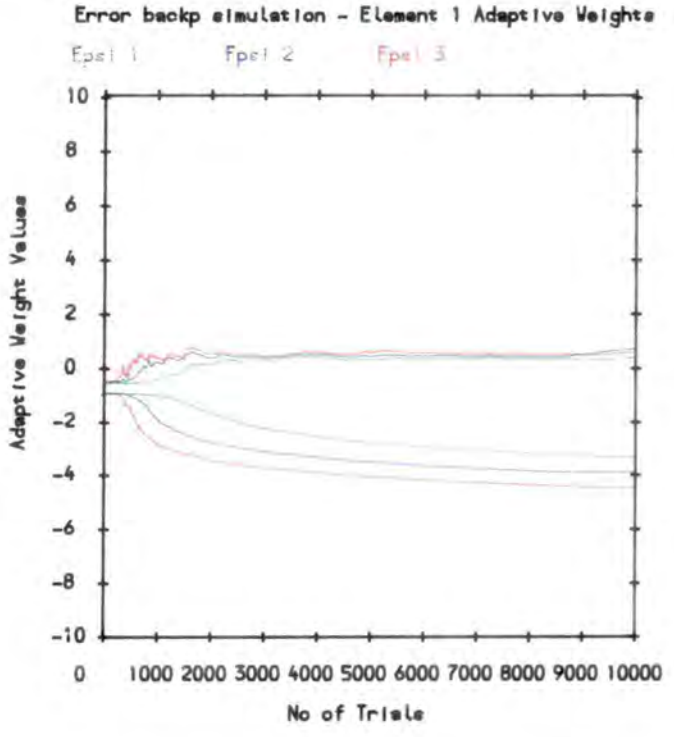


Figure 7.25b - w_{111}, w_{112} vs Number of Trials

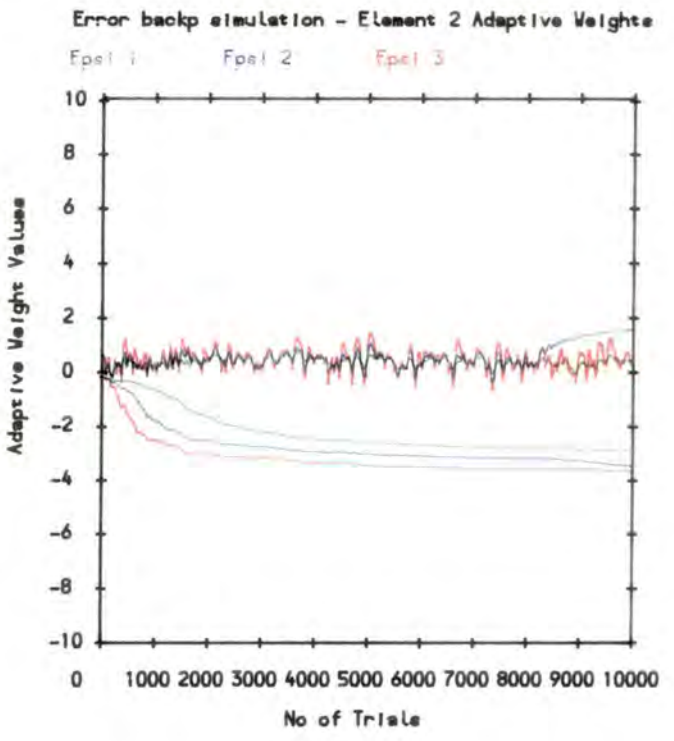


Figure 7.25c - w_{211}, w_{212} vs Number of Trials

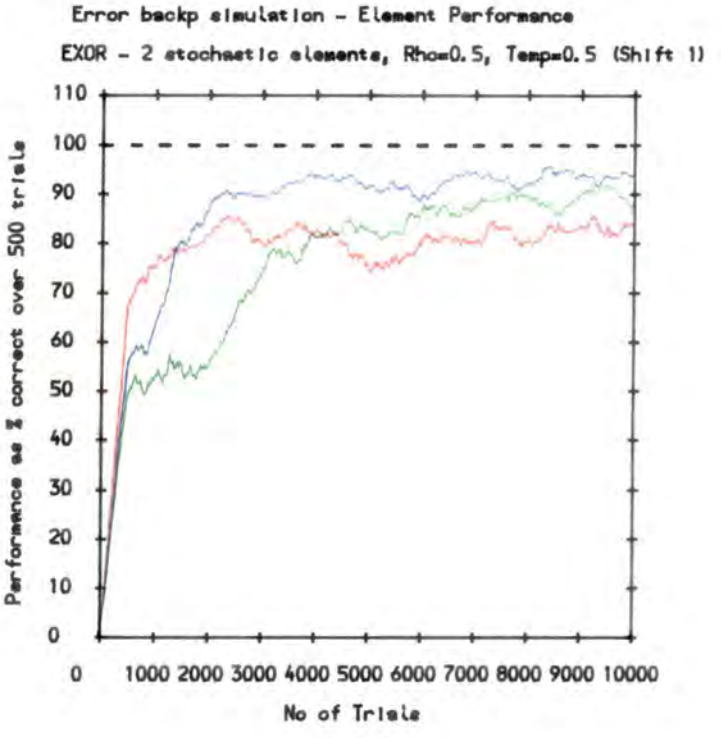


Figure 7.26a - % performance vs Number of Trials (Shift 1)

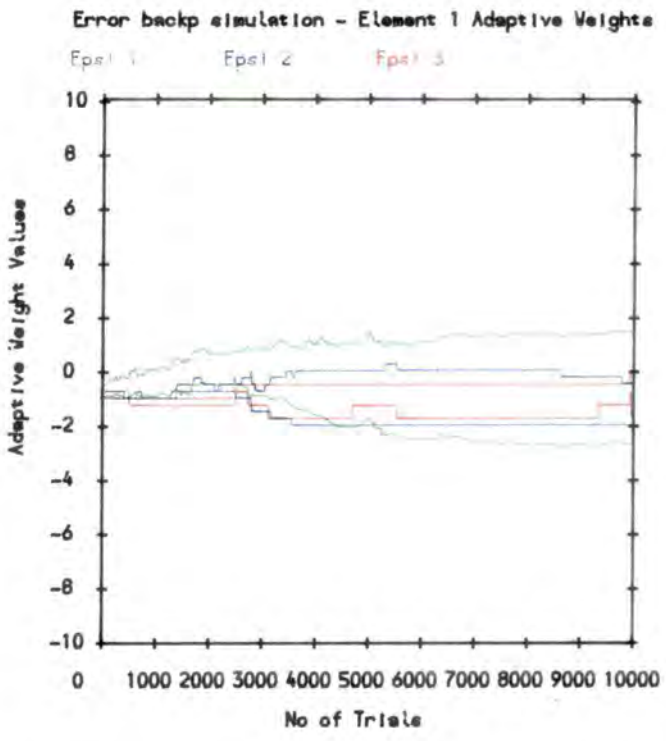


Figure 7.26b - w_{111}, w_{112} vs Number of Trials (Shift 1)

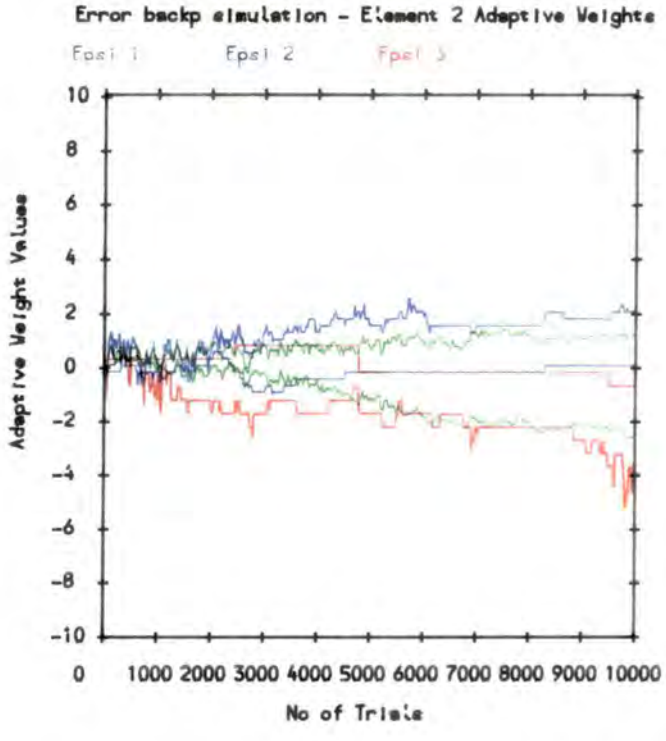


Figure 7.26c - w_{211}, w_{212} vs Number of Trials (Shift 1)

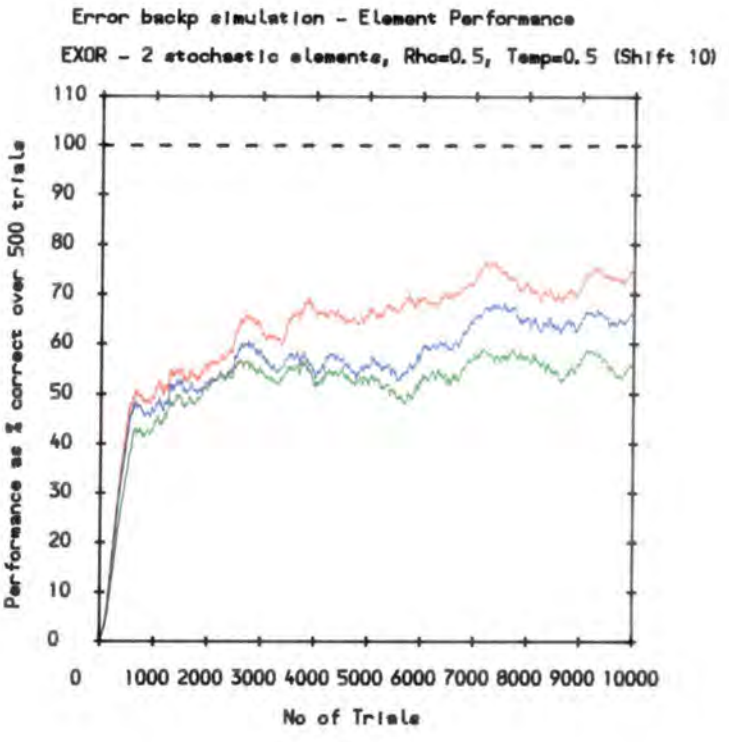


Figure 7.27a - % performance vs Number of Trials (Shift 10)

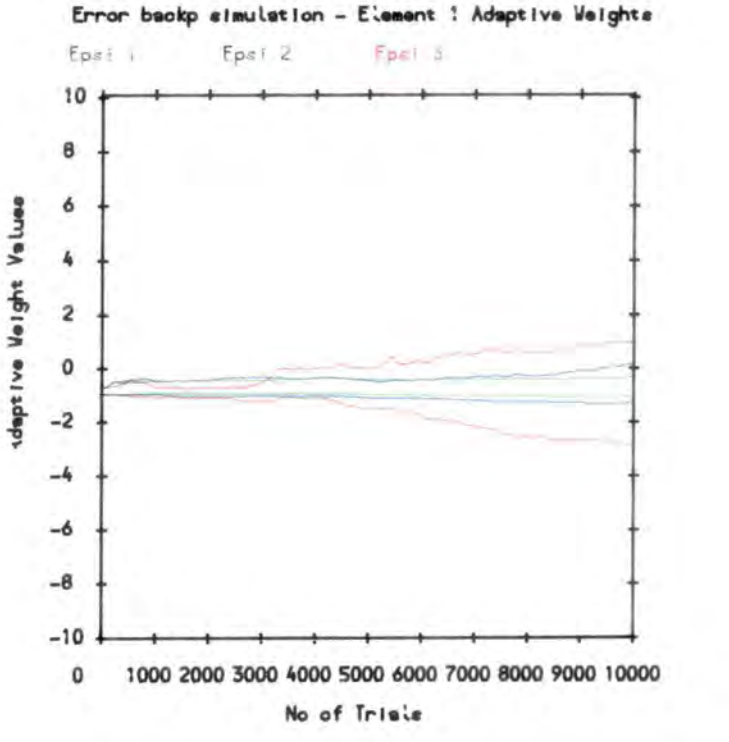


Figure 7.27b - w_{111}, w_{112} vs Number of Trials (Shift 10)

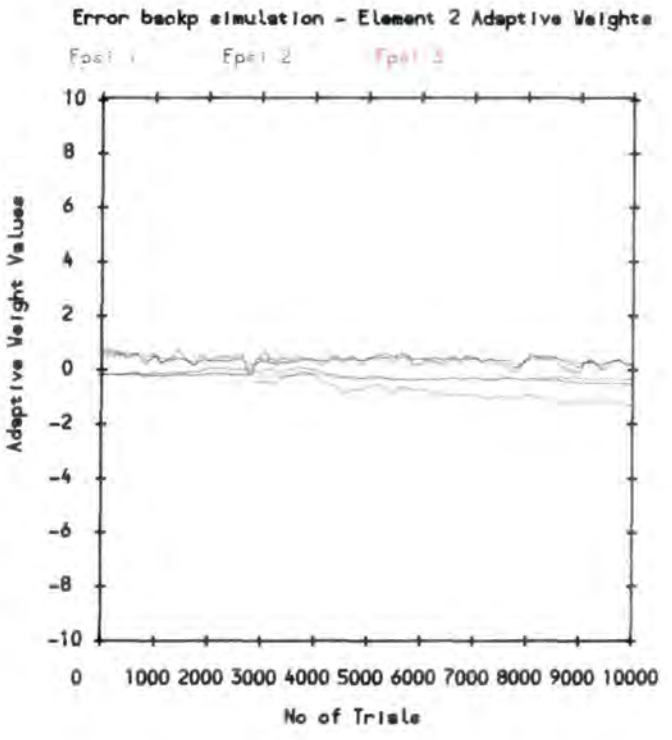


Figure 7.27c - w_{211}, w_{212} vs Number of Trials (Shift 10)

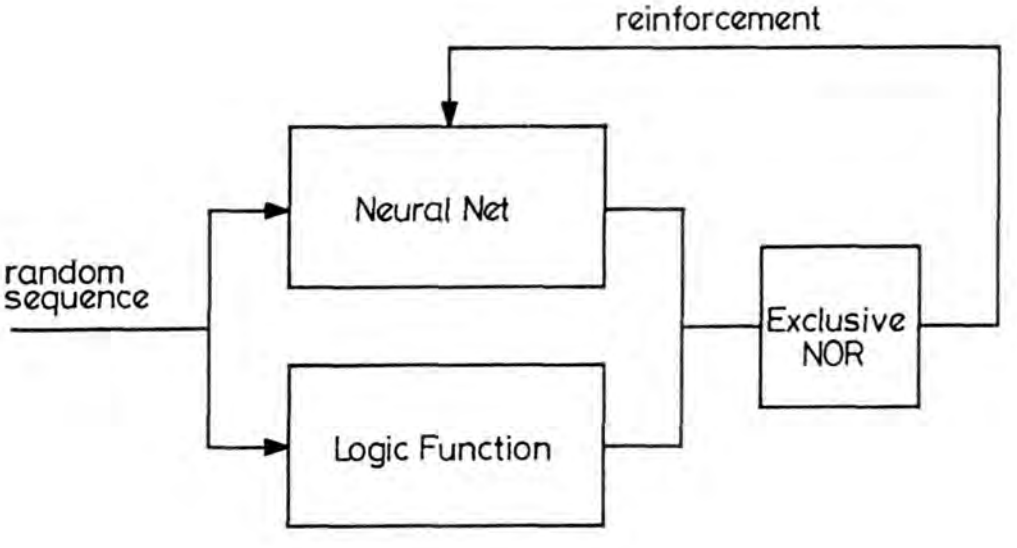


Figure 7.28 Method for training a reinforcement learning neural network

Table 7.1 - Reward Probabilities - Example 1

x	d(x, +1)	d(x, 0)
1,0	0.1	0.9
1,1	0.9	0.1

Table 7.2 - Reward Probabilities - Example 2

x	d(x, +1)	d(x, 0)
0,0	0.1	0.9
0,1	0.9	0.1
1,0	0.9	0.1
1,1	0.1	0.9

Table 7.3 - Error backpropagation - one element problem

Input	Output
1,0	0
1,1	1

Table 7.4 - Exclusive OR truth table

Input	Output
0,0	0, 0
0,1	0, 1
1,0	0, 1
1,1	0, 0

Chapter Eight

Comparative Performance Evaluations

8.1 Introduction

The previous chapters have presented the basic theory and small-scale simulations of A_{R-P} and error backpropagation networks using various learning rule modifications to try to obtain improved performance. A performance guide, based on the percentage of input/output vectors correctly learned over a moving window of 500 trials was introduced as a means of comparing these fundamentally different networks.

The Durham Neural Network simulation program, described in Appendix 4, is an experimental program, developed in stages in order to simulate and understand these networks in a variety of small-scale configurations. It was considered that the best way to compare these different learning methods was to study the learning rate of each method over a sufficient number of independent runs, for a number of different yet taxing examples and to draw conclusions from these.

This chapter presents a series of comparative studies of A_{R-P} learning automata against deterministic error backpropagation for identical problems, using suitable parameters. The simulation program was designed to evaluate a 3 dimensional network of elements, X elements long, Y elements wide and Z elements deep. Each level therefore contained the same number

of elements as the level above it. This was in contrast to other network simulations in the literature, [2], [5], which usually have a much larger 'hidden' layer. As explained in Chapter Five, the number of elements in the 'hidden' layer limits the number of separate input–output vector combinations for a particular network size, which may be learned. Implementing a restricted array size as performed in these simulations, aside from a uniform and computationally convenient approach, allowed the introduction of potential local minima into the problem, enabling interesting comparisons to be made between different learning algorithms. Learning rate histograms are used to display the performance of each method.

8.2 Data Collection

The terms used in the following sections follow the convention described in Chapter Seven. The data acquisition was performed for a total of 2000 runs ie. executions of the simulation program, each consisting of 100,000 trials and commencing with initially randomised weights. At each trial, the percentage correct performance criterion described in the last chapter was used to observe whether the array had completed the learning process. If the array had successfully learned, then the trial number at this point was stored in a data file and a new run begun. If the array had not learned by trial 100,000 then this figure was stored in the data file. This should not be taken as indicating that the array could not solve the problem, only that it had not done so by this time and thereby provide an upper bound to the training. The stored values in the data file, were subsequently used

to construct a histogram. The X axis of the histogram depicted 'bins' of width 500, covering the trial range 0 to 100,000 with the Y axis represent the number of program runs which fell into these bounds. In this way, a visual measure of the learning process was obtained for the various types of network. The histograms were plotted individually to enable separate scaling, and also together for each type of problem to enable convenient comparison.

Due to the inherent differences between modes of learning for the types of network considered, no one set of suitable parameters could be chosen for both. Furthermore, the choice of parameter describing the step size of the weight update could be very important, particularly for stochastic implementations and is application dependent. Clearly, coarse values in some cases may be appropriate, enabling relatively large weight changes and fast gradient descent for simple problems, yet inappropriate for the finer changes which might be required for more involved problems requiring many more elements. As explained in Chapters Five and Six, the A_{R-P} algorithm uses both a large and a small weight update factor in the learning process enabling both fast yet coarse change and small but slow 'fine tuning' whereas the error backpropagation method uses a factor proportional to the differential error relative to the weights to form the weight updates together with a 'momentum' term which takes into account past weight updates in order to speed the gradient descent.

It was decided to select the best set of parameters from the simple simulation results of Chapter Six and Chapter Seven which were themselves selected from many more variations performed during the course of this

research. The parameters considered most appropriate for comparison of the A_{R-P} networks were reward probability = 0.9, punish probability = 0.1, ρ = 0.5, λ = 0.01 and Temperature T = 0.5. Similarly, the parameters considered most appropriate for the error backpropagation networks, from the earlier simulations, were ϵ = 0.5, α = 0.5 and Temperature, T = 0.5. The setting for the general weight updating constant ρ for the A_{R-P} network may be considered similar to the setting for the general weight updating constant α for the error backpropagation network.

8.3 Exclusive OR problem

The first problem chosen for comparison, was the Exclusive OR problem, which is the simplest case of an even parity check and which was explained in Chapter Five to be a non-trivial problem which cannot be solved by single layer arrays.

8.3.1 2x1x2 topology

The 2 x 1 x 2 element topology used for simulation is shown in Figure 8.1 and the truth table in Table 8.1. The performance histogram for the A_{R-P} network using success/failure reinforcement is shown in Figure 8.2a and using gradient reinforcement in Figure 8.2b. The error backpropagation histogram is shown in Figure 8.2c and all plots together in Figure 8.2d. These results may be interpreted as follows.

Consider the histogram shown in Figure 8.2a. It may be seen that the A_{R-P} array using success/failure reinforcement rapidly learns the correct

response under 100000 trials in all but approximately 47 runs. The histogram peaks between 5000 and 5500 trials indicating where the network successfully passed the learning criterion during 72 runs of the program. The majority of other successful runs cluster about this peak gradually dying away towards the 100000 trial mark. The sharp rise at the 100000 trial mark is a boundary effect, implying those runs which might have completed after this time and does not represent an anomaly in the curve.

In comparison, consider the histogram shown in Figure 8.2b. In this case, the A_{R-P} network is using gradient reinforcement in order to learn to solve the problem. It may be seen that whilst there is a peak where 76 runs successfully passed the learning criterion between 6000 and 6500 trials, there is a much larger rise at the final 100000 trial mark, where approximately 163 runs failed to complete learning by this time. Figure 8.2c shows the histogram for error backpropagation solving the same example. In this case, there is a peak where 151 runs successfully passed the learning criterion between 7000 and 7500 trials but approximately 698 runs failed to complete by the final trial mark. Before commenting on this case, consider Figure 8.2d which depicts all three histograms on the same set of axes. It may be seen that the peak due to error backpropagation is very narrow and defined whereas those due to the A_{R-P} methods are much more spread out although all three peaks are relatively close.

The conclusion which may be drawn, taking all three methods into account, is that A_{R-P} arrays using success/failure reinforcement appears very successful in solving the non-trivial Exclusive OR problem. The A_{R-P}

network using gradient descent is slightly less successful since more runs fail to complete by the final trial mark but otherwise the curves are similar. However, roughly 30% of the error backpropagation runs do not complete by the 100,000th trial. It is considered that this is due to the presence of local minima trapping the error gradient descent. Although the A_{R-P} network example also uses gradient descent, the element adaptation is probabilistic which allows it to jump out of this minima. To test this hypothesis, the simulation was repeated using the network shown in Figure 8.3. This was exactly the same in operation as the network shown in Figure 8.1 except that each layer had an extra element and the additional input and output was set to zero. This introduced extra capacity into the network which results in a higher dimensional search space which may remove the minima.

8.3.2 3x1x2 topology

The performance histogram for the A_{R-P} network with the topology shown in Figure 8.3 using success/failure reinforcement is shown in Figure 8.4a and using gradient reinforcement in Figure 8.4b. The error backpropagation histogram is shown in Figure 8.4c and all plots together in Figure 8.4d. These results may be interpreted as follows.

Consider the histogram shown in Figure 8.4a. It may be seen that the A_{R-P} array using success/failure reinforcement again rapidly learned the correct response under 100000 trials in all runs. The histogram peaks between 2500 and 3000 trials indicating where the network successfully passed the learning criterion during 122 runs of the program. As before, the majority

of other successful runs cluster about this peak gradually dying away towards the 100000 trial mark although this time, all runs complete before the 100000 trial boundary.

In comparison, consider the histogram shown in Figure 8.4b where the A_{R-P} network is using gradient reinforcement. In this case, there is a peak where 105 runs successfully passed the learning criterion between 2500 and 3000 trials and again all runs complete before the final trial boundary. Figure 8.4c shows the histogram for error backpropagation solving the same example. In this case, there is a peak where 184 runs successfully passed the learning criterion between 6500 and 7000 trials but this time only 125 runs failed to complete by the final trial mark. Consider Figure 8.4d which depicts all three histograms on the same set of axes.

The conclusion which may be drawn, is that A_{R-P} arrays using success/failure reinforcement and using the gradient descent method, is very similar in this larger topology case. Furthermore, error backpropagation is much more successful and this time, only about 6% of the backpropagation runs fail to complete by the final trial mark. Therefore the addition of extra elements has improved performance by increasing the dimensionality and therefore the search space of the problem.

8.4 Parity check problem

The Exclusive OR problem can be extended into general parity checking. This problem is one which is extremely difficult to generalise from an incomplete training set, the proof of which was given in the study of

Perceptrons by Minsky and Papert, [92]. The parity problem here involves the examination of an n bit input vector. If there are an odd number of 'on' bits in the vector, then an output parity bit must be generated, 'on' and appended to make an even n+1 bit vector. If there are an even number of 'on' bits in the vector, then an output parity bit, 'off' must be generated and appended to make an even n+1 bit vector. The 5 x 1 x 3 topology used for simulation is shown in Figure 8.5 and the truth table in Table 8.2. The performance histogram for the A_{R-P} network using success/failure reinforcement is shown in Figure 8.6a and using gradient reinforcement in Figure 8.6b. The error backpropagation histogram is shown in Figure 8.6c and all plots together in Figure 8.6d. These results may be interpreted as follows.

Consider the histogram shown in Figure 8.6a. It may be seen that the A_{R-P} array using success/failure reinforcement failed to learn the correct response in under 100,000 trials in all but 3 runs. Similarly, as shown in Figure 8.6b, the A_{R-P} network using gradient reinforcement was unable to learn the correct response in under 100,000 trials in all but 33 runs. In contrast, error backpropagation was able to learn to solve the problem in 404 runs (roughly 20%), as shown in Figure 8.6c.

This result was expected since the parity problem is one of the hardest problems to learn, since it requires a structure which discriminates between odd and even. It is hard to learn, given a complete truth table for a particular length vector particularly with the relatively few elements in the topology used and very difficult to generalise from an incomplete training set.

It would be interesting to repeat the simulations with many more elements to observe any change in performance.

8.5 Reversal problem

The reversal problem, [109], involved supplying the array with an input vector, the task being to output the same vector but reversed in order. The 5 x 1 x 3 topology used for simulation is shown in Figure 8.7 and the truth table in Table 8.3. The performance histogram for the A_{R-P} network using success/failure reinforcement is shown in Figure 8.8a and using gradient reinforcement in Figure 8.8b. The error backpropagation histogram is shown in Figure 8.8c and all plots together in Figure 8.8d. These results may be interpreted as follows.

Consider the histogram shown in Figure 8.8a. It may be seen that the A_{R-P} array using success/failure reinforcement learned the correct response under 100000 trials in approximately 35 % of the runs and approximately 1302 runs failed to complete by the final trial. This was considered to be due to the limitation of the global reinforcement for a relatively large input–output vector.

In contrast the histogram shown in Figure 8.8b for the A_{R-P} network using gradient reinforcement shows a peak of 70 runs between 12000 and 12500 trials with other close peaks around this. All runs complete before the final trial mark. Finally consider Figure 8.8c, which shows the histogram for error backpropagation solving the same example. This figure too shows a peak where 263 runs were successful between 6000 and 6500 trials with other

close peaks in the range 5500 to 8500 trials. Once again, all runs successfully passed the learning criterion before the final trial mark. Figure 8.8d shows that the error backpropagation learning histogram is narrow about the peak than the more spread-out A_{R-P} using gradient reinforcement.

In conclusion, this example shows the limitations of the A_{R-P} arrays using success/failure reinforcement and the advantage of recoding the reinforcement signal using the gradient method. Error backpropagation appears superior in this example, although the hardware and connection advantages of the A_{R-P} arrays as compared to those of error backpropagation should be recalled.

8.6 Zero Image problem

This task involved the array being supplied with an input vector representing the binary image of a zero, the array being required to output the same vector. Whilst this task may seem trivial, it was originally devised to examine the capability of feed-forward networks in a feedback configuration to reconstruct degraded inputs. This is explained further in Appendix 4. The 5 x 5 x 2 topology used for simulation is shown in Figure 8.9. The performance histogram for the A_{R-P} network using success/failure reinforcement is shown in Figure 8.10a and using gradient reinforcement in Figure 8.10b. The error backpropagation histogram is shown in Figure 8.10c and all plots together in Figure 8.10d. These results may be interpreted as follows.

Consider the histogram shown in Figure 8.10a. It may be seen that the A_{R-P} array using success/failure reinforcement completely failed to learn

the correct response before the final trial mark. There are over 33 million combinations of a 25 bit vector and success/failure reinforcement was useless in this case.

Use of gradient reinforcement, as shown in Figure 8.10b was much better. The array solved the problem roughly 50% of the time before the final trial mark although with no obvious peak. In vivid contrast, error backpropagation as shown in Figure 8.10c solved the problem easily with 1122 runs completing between 500 and 1000 trials. Recall that the averaging procedure is taken over a 500 trial window, so this represents very fast learning. Most of the other runs completed around this peak. A few trail off with only approximately 10 runs failing to complete by the final trial mark. This is considered to be due to the original random setting of the weights introducing a local minimum effect into the error descent path. Figure 8.10d shows the relative performances.

In summary, this problem, whilst not the most complicated, is the largest attempted during this thesis. As previously explained, small problems were considered the best in order to cross-evaluate different learning methods, but this case indicates the need to develop better reinforcement learning methods for the A_{R-P} networks or hybrid approaches involving both A_{R-P} and error backpropagation types of element to enable larger networks and therefore more interesting problems to be examined.

8.7 Channel Encoder problem

The channel encoder problem, [72], as explained in Chapter Six, involved supplying the array with an n-bit vector with the restriction that only one bit may be 'on'. Therefore there are only n possible input vectors. The array was required to output the input vector, but there was an additional problem in that the number of elements comprising the middle or 'hidden' layer was only $\log_2(n)$. This therefore required the array to develop the optimum codes to communicate the input vector across the bandwidth constriction. Where n=4, the minimum number of 'hidden' elements is 2, and the problem is known as the 4–2–4 channel encoder problem. For n=8, the minimum number of 'hidden' elements is 3 and the problem is known as the 8–3–8 channel encoder problem.

8.7.1 4–2–4 Channel Encoder problem

The 4 x 1 x 3 topology used for simulation is shown in Figure 8.11 and the truth table in Table 8.4. The performance histogram for the A_{R-P} network using success/failure reinforcement is shown in Figure 8.12a and using gradient reinforcement in Figure 8.12b. The error backpropagation histogram is shown in Figure 8.12c. All plots are shown in Figure 8.12d. These results may be interpreted as follows.

Consider the histogram shown in Figure 8.12a. It may be seen that the A_{R-P} array using success/failure reinforcement learned the correct response in roughly 40% of the cases under 100000 trials without any peak appearing. Approximately 1159 runs failed to complete before the 100000 trial mark.

In comparison, consider the histogram shown in Figure 8.12b where the A_{R-P} network was using gradient reinforcement in order to learn to solve the problem. In this case, there were 96 runs where the A_{R-P} network successfully completed the problem between 7000 and 7500 trials with most of the other peaks clustering around. The A_{R-P} network completed all runs before the final trial mark. Finally consider Figure 8.12c which shows the histogram for error backpropagation solving the same example. In this case, there is a peak where 247 runs successfully passed the learning criterion between 5500 and 6000 trials with the other peaks clustered closely around. Again the network completed all runs before the final trial mark. Figure 8.10d enables comparison of all three plots, clearly showing a narrower peak for error backpropagation compared with A_{R-P} using gradient reinforcement.

In conclusion, this is a small-scale but unusual problem, in that the 'hidden' elements were deliberately restricted in order to force the network into developing an optimum coding scheme in order for success. Gradient reinforcement clearly increases the A_{R-P} networks success rate but error backpropagation appears more successful at solving the problem. Increasing the size of the encoder problem from n=4 to n=8 makes the problem much more difficult and the relative performances of each method were investigated for this more demanding case.

8.7.2 8–3–8 Channel Encoder problem

This extends the 4–2–4 encoder problem to n=8 using 3 'hidden' elements. The 8 x 1 x 3 topology used for simulation is shown in Figure

8.13 and the truth table in Table 8.5. The performance histogram for the A_{R-P} network using success/failure reinforcement is shown in Figure 8.14a and using gradient reinforcement in Figure 8.14b. The error backpropagation histogram is shown in Figure 8.14c. All plots are shown together in Figure 8.14d. These results may be interpreted as follows.

Consider the histogram shown in Figure 8.14a. It may be seen that the A_{R-P} array using success/failure reinforcement completely failed to learn the correct response before the final trial mark. The A_{R-P} array using gradient reinforcement solved the problem during some 50% of the runs with approximately 1055 runs failing to complete before the final trial mark as shown in Figure 8.14b. In contrast, error backpropagation solved the problem between 9000 and 11,500 trials as shown in Figure 8.14c.

In conclusion, the A_{R-P} array using success/failure reinforcement fails to solve the problem. The use of gradient reinforcement improves the situation but error backpropagation is by far, the most successful method.

8.8 Conclusions and Summary – Chapter Eight

It is possible to conclude from these results that firstly error backpropagation was a superior method to A_{R-P} learning in terms of speed although the method did get caught in apparent local minima when the number of hidden elements was restricted as shown by the different size networks applied to the Exclusive OR problem. A_{R-P} learning scaled poorly, even when using gradient reinforcement, due to the restrictive nature of the global reward signal although it did not get caught in local minima to the same extent due to

the probabilistic nature of the reinforcement. Error backpropagation appeared better suited to solving larger problems. These simulations encourage the development of hybrid schemes which use the best features of each approach to allow the networks to learn. In particular, methods which incorporate the hardware, noise and connection advantages of stochastic implementations are particularly worth examining and these are discussed as future work in the concluding chapter.

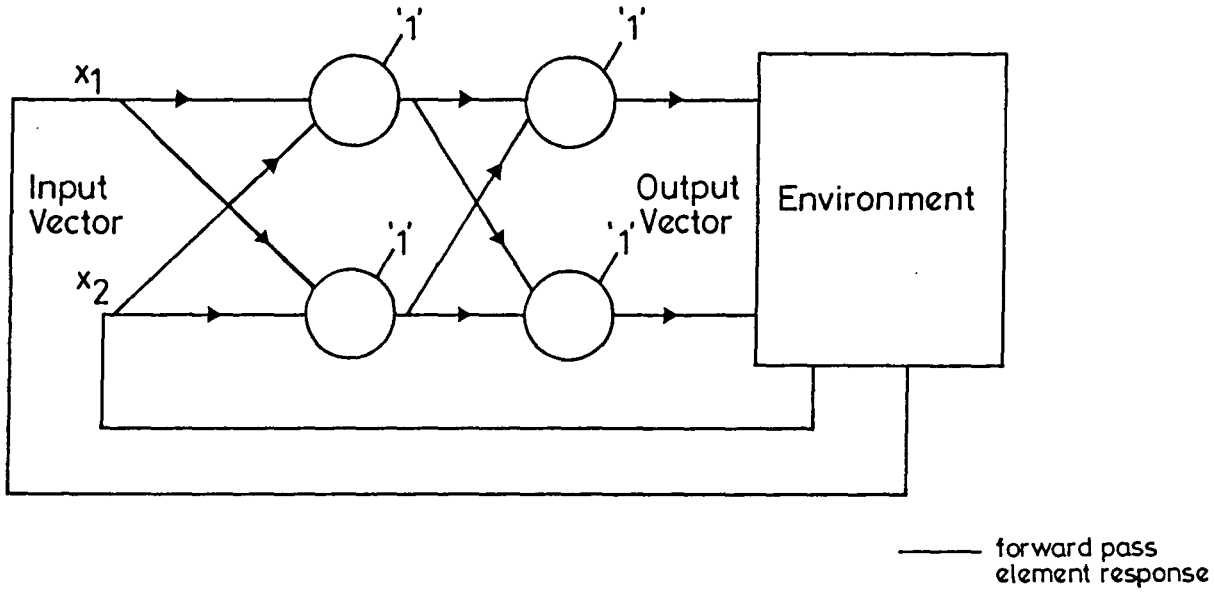


Figure 8.1 Exclusive OR topology - 2 x 1 x 2

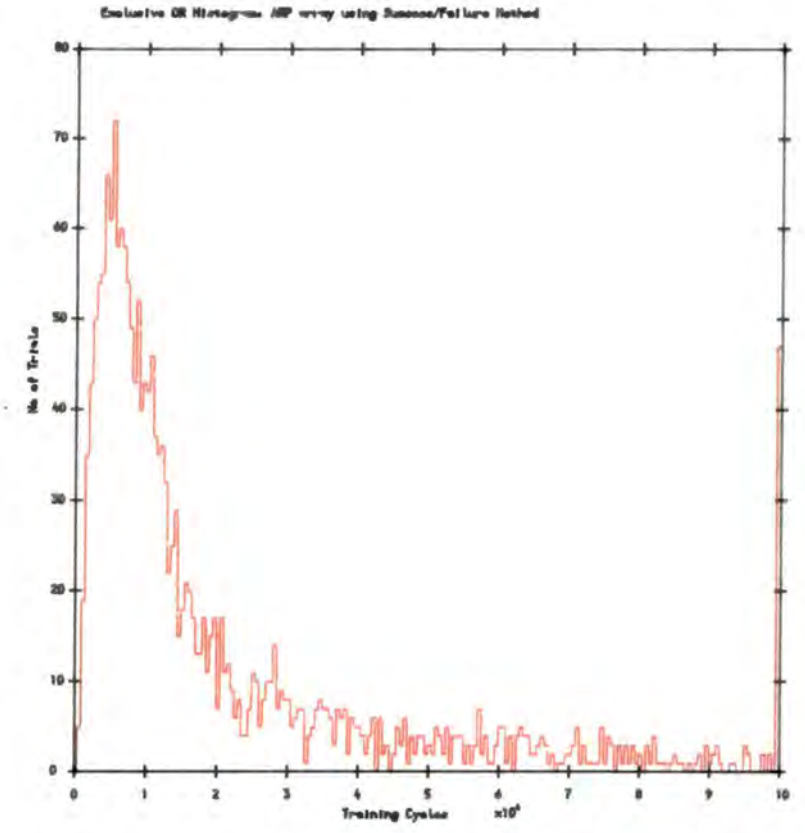


Figure 8.2a Exclusive OR Histogram using A_{R-P} Success/Failure method

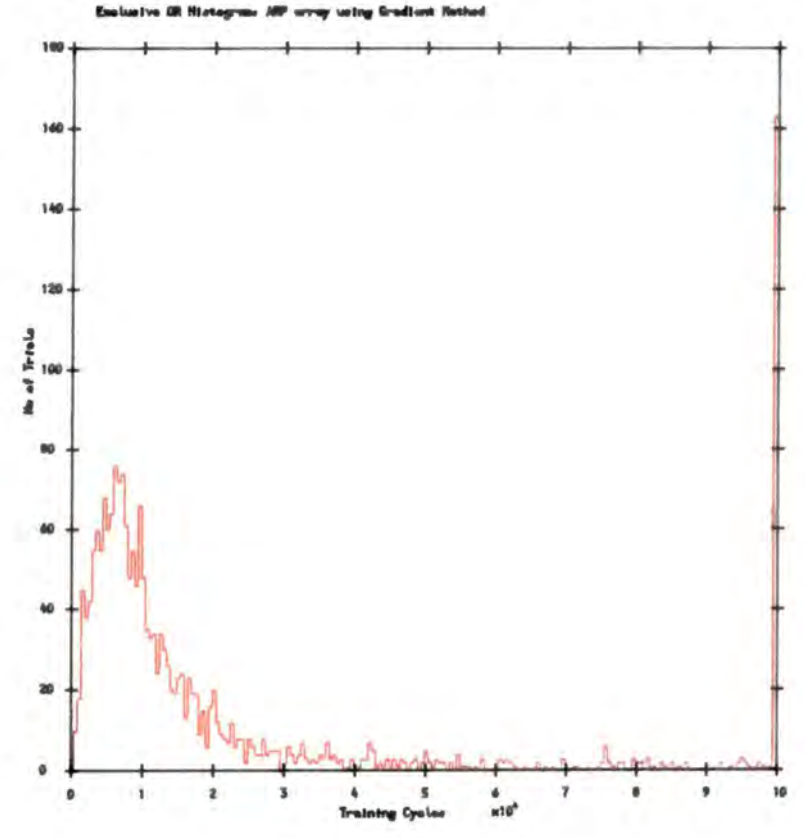


Figure 8.2b Exclusive OR Histogram using A_{R-P} Gradient method

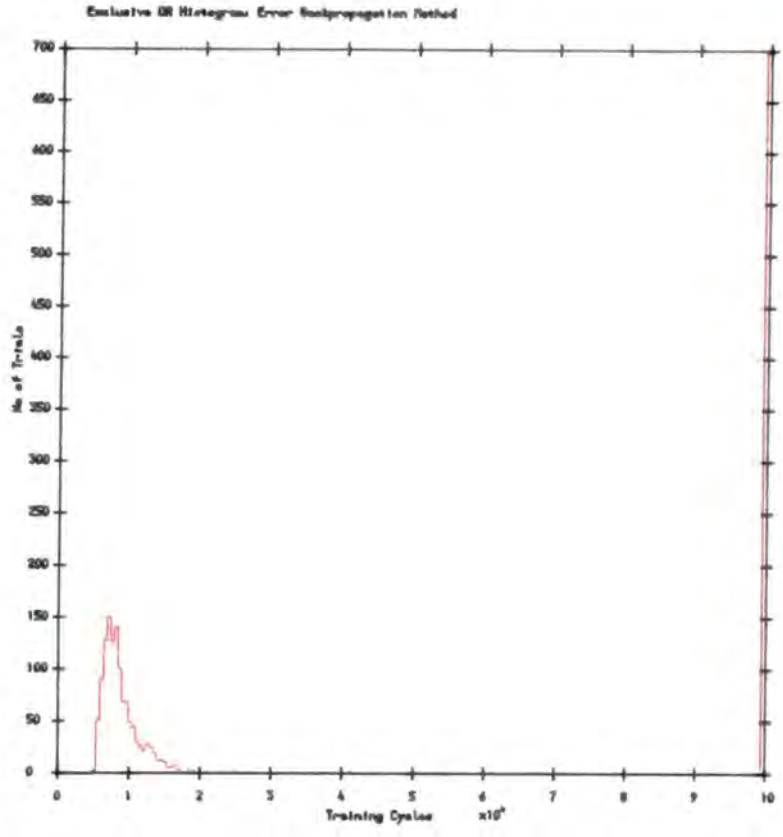


Figure 8.2c Exclusive OR Histogram using error backpropagation

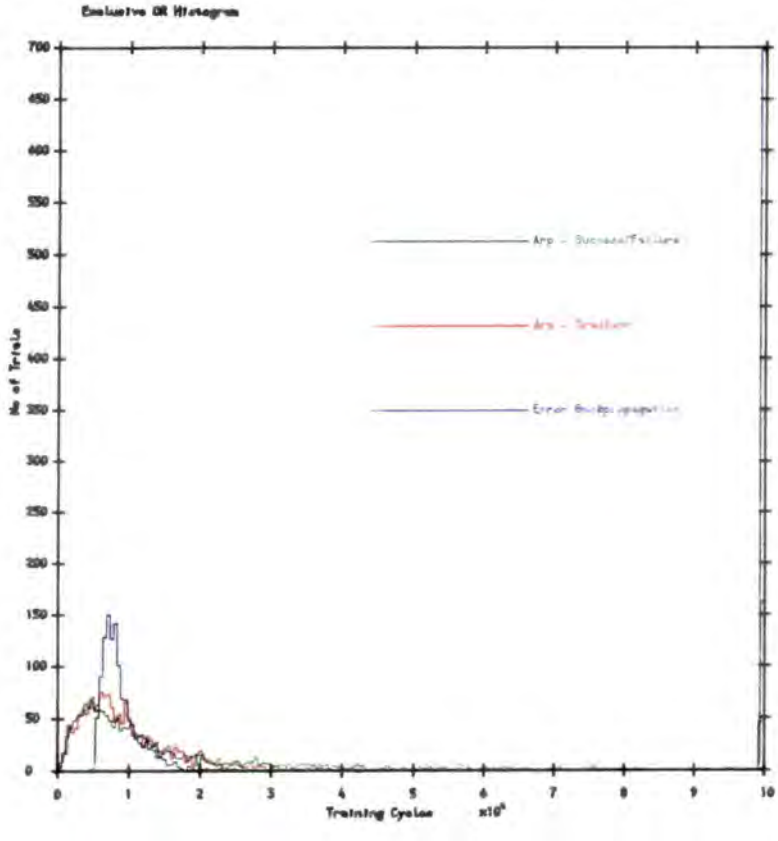


Figure 8.2d Exclusive OR Histogram (all plots)

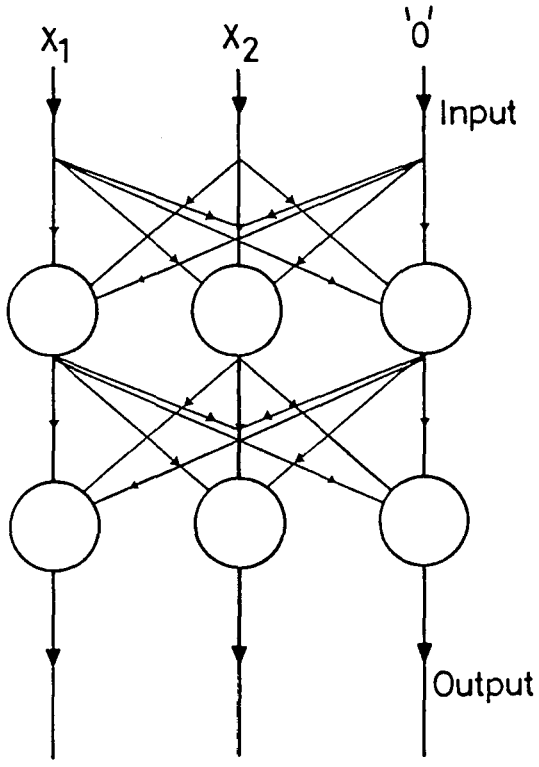


Figure 8.3 Exclusive OR topology - 3 x 1 x 2

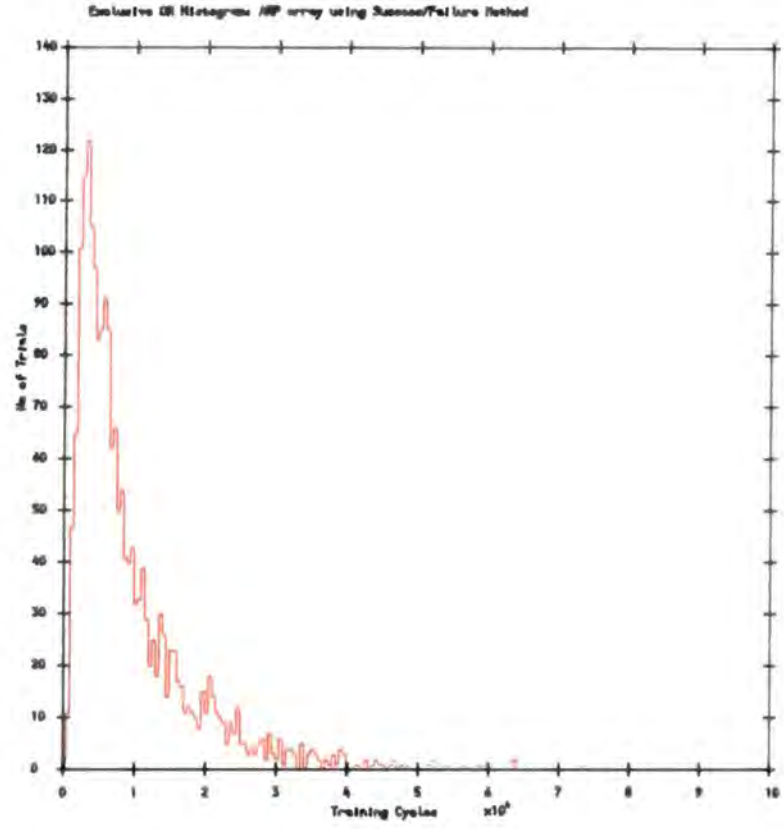


Figure 8.4a Exclusive OR Histogram using A_{R-P} Success/Failure method

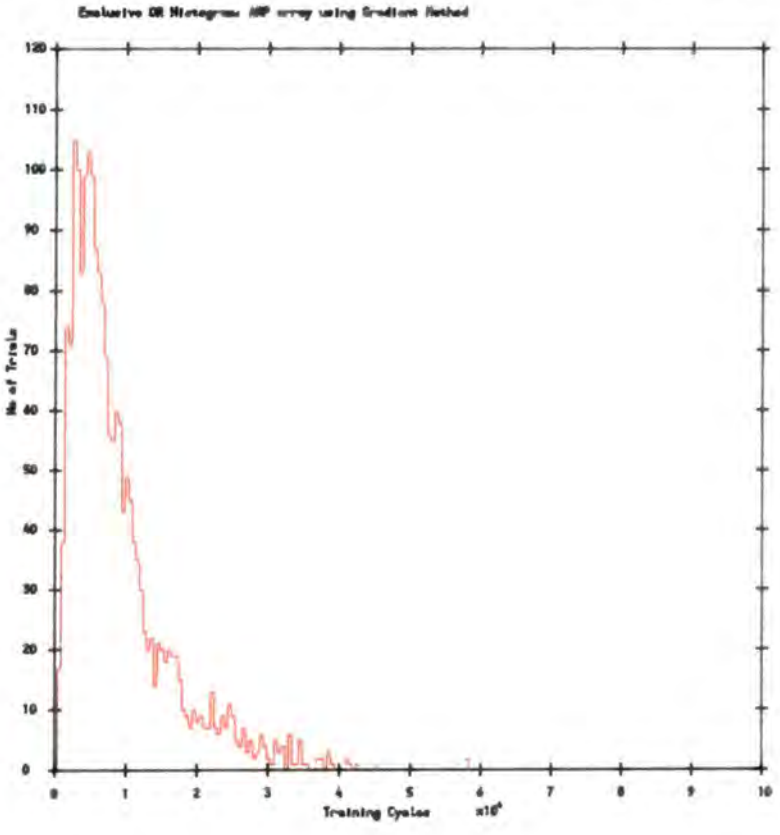


Figure 8.4b Exclusive OR Histogram using A_{R-P} Gradient method

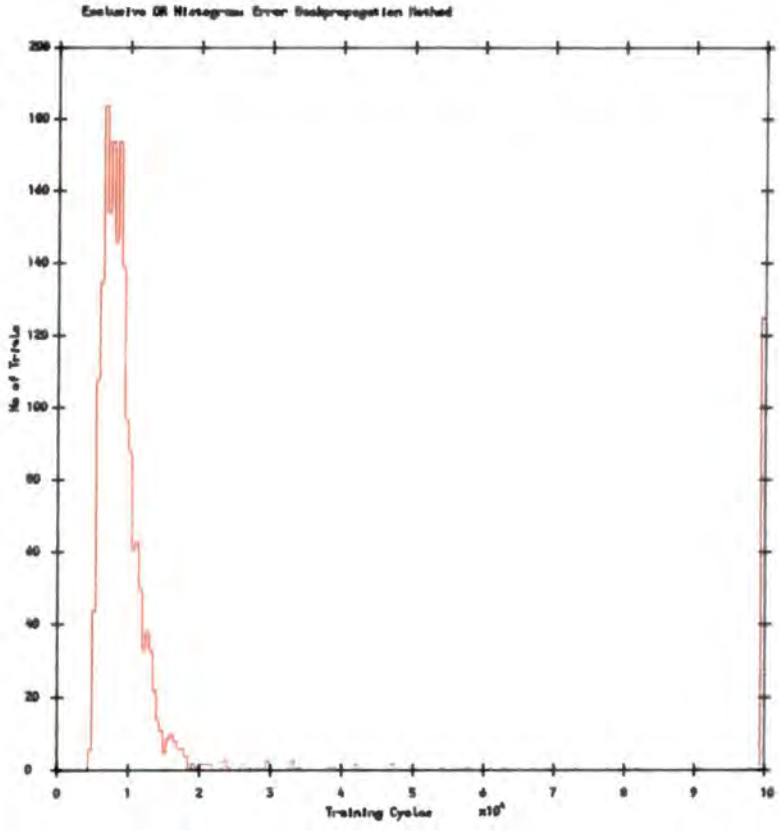


Figure 8.4c Exclusive OR Histogram using error backpropagation

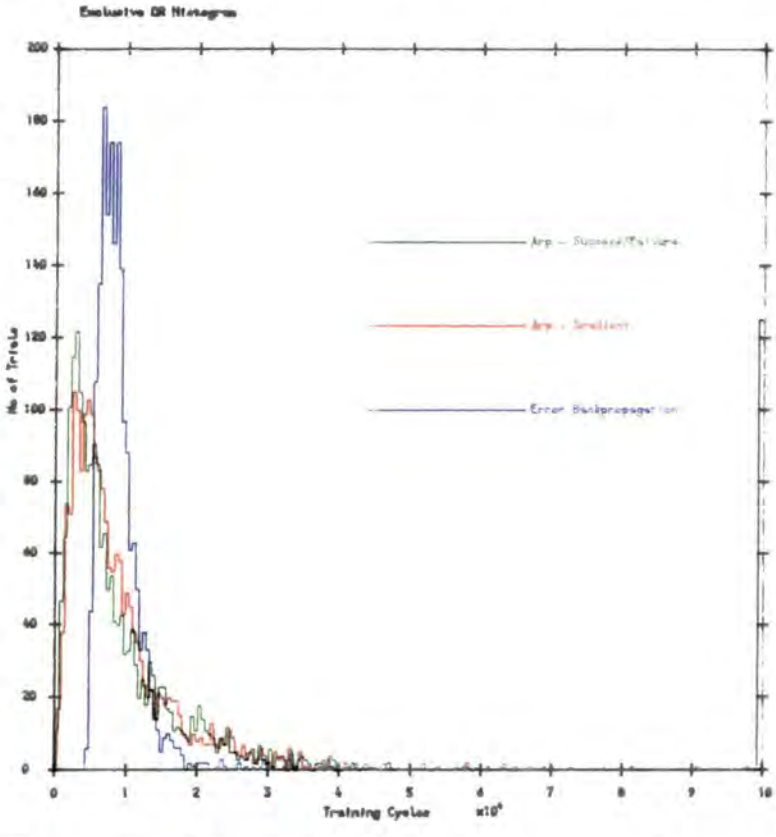


Figure 8.4d Exclusive OR Histogram (all plots)

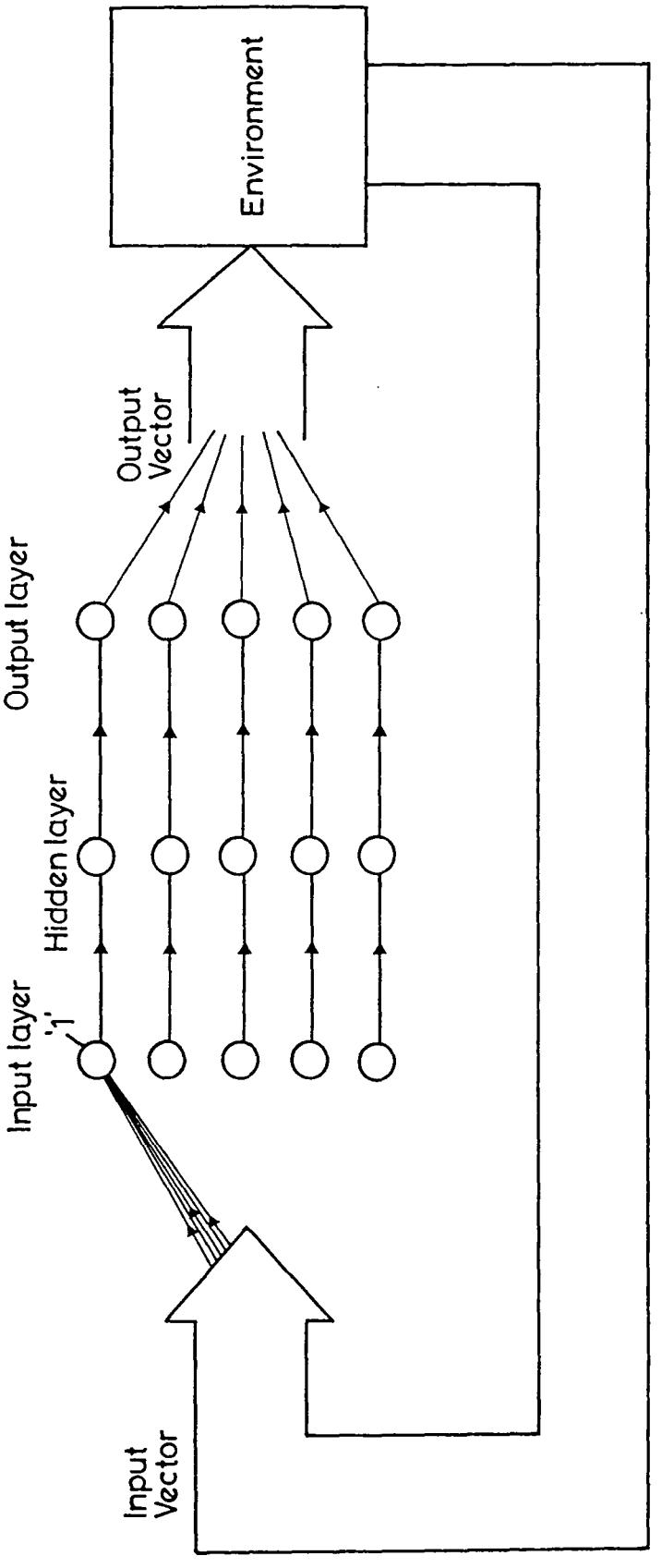


Figure 8.5 Parity check topology

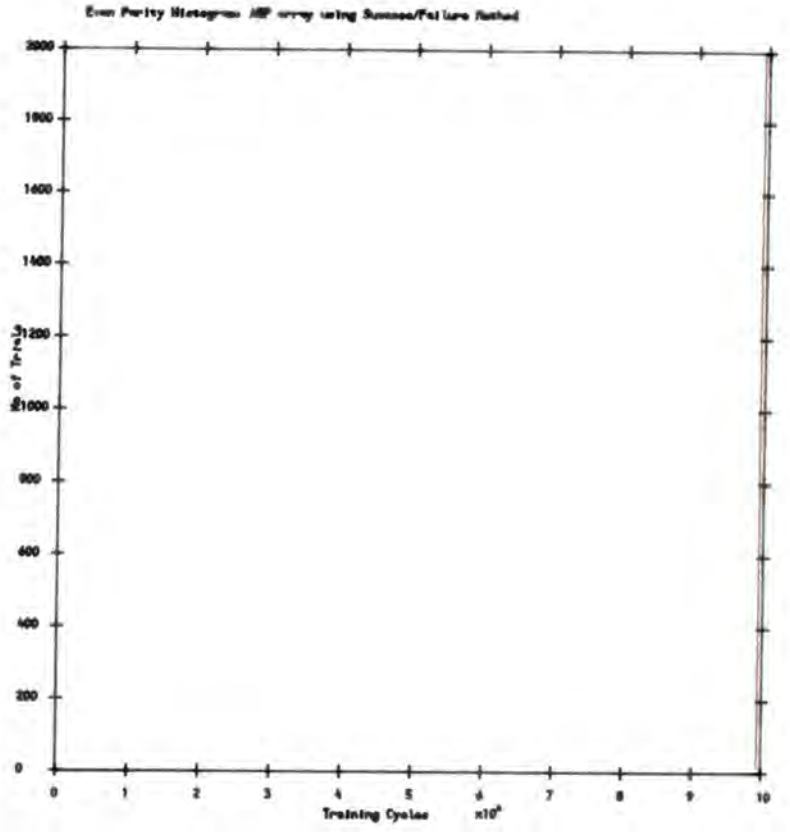


Figure 8.6a Parity check Histogram using A_{R-P} Success/Failure method

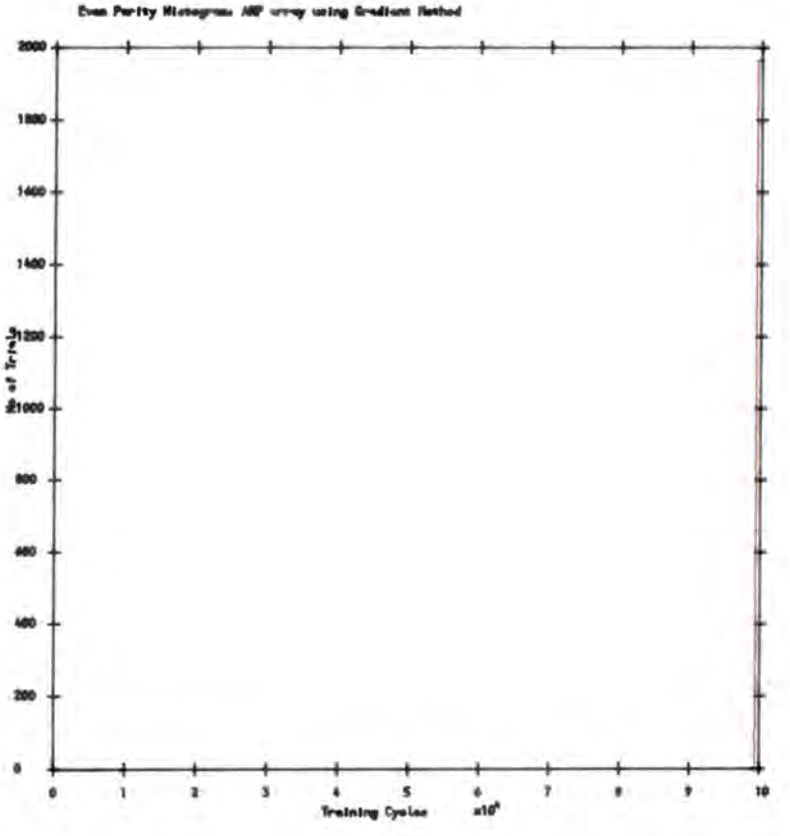


Figure 8.6b Parity check Histogram using A_{R-P} Gradient method

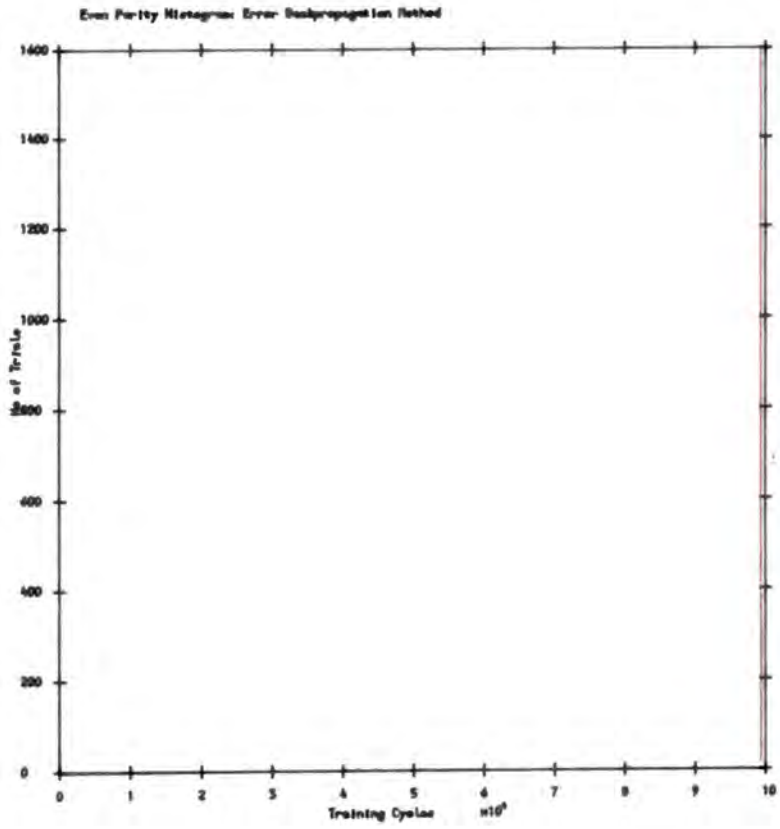


Figure 8.6c Parity check Histogram using error backpropagation

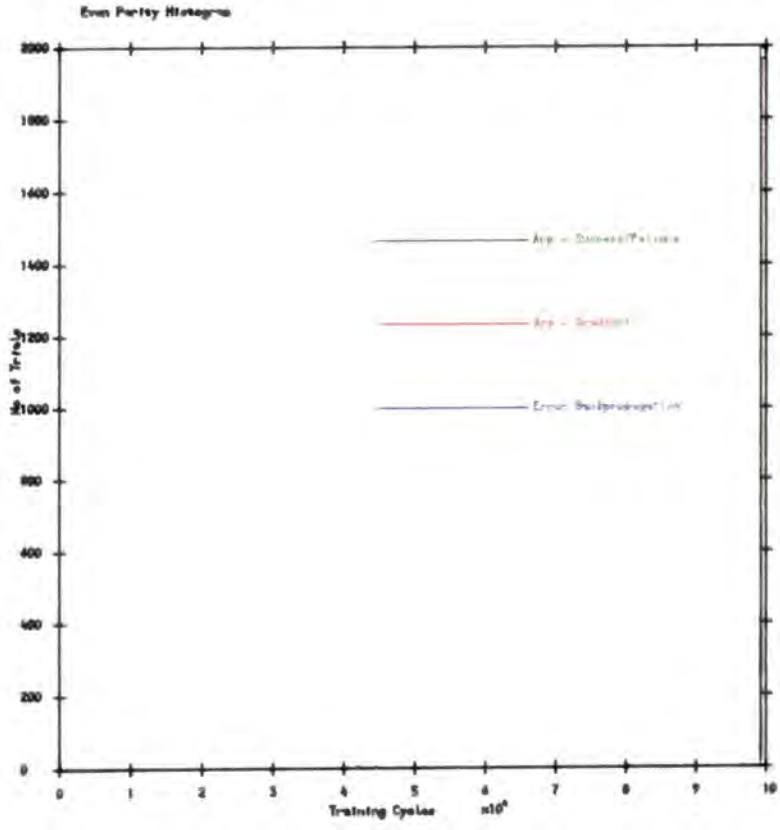


Figure 8.6d Parity check Histogram (all plots)

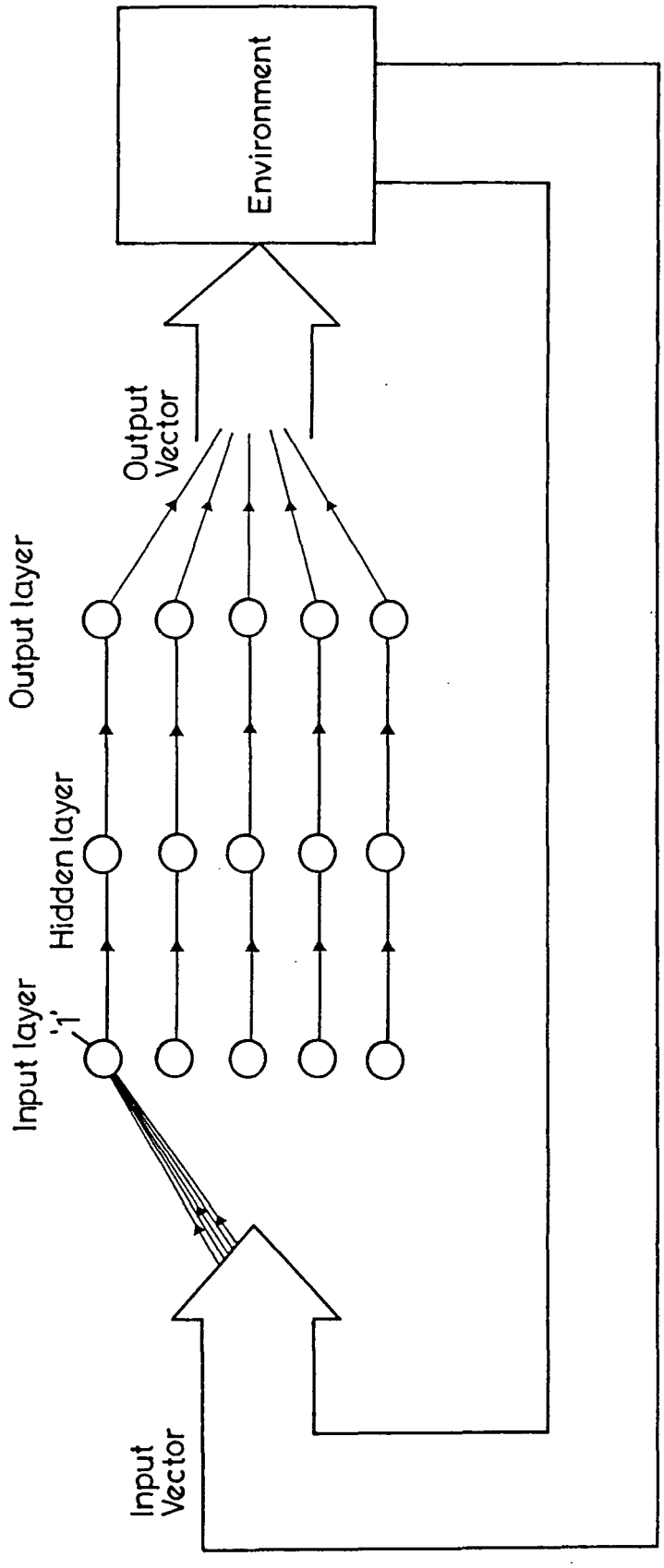


Figure 8.7 Reversal topology

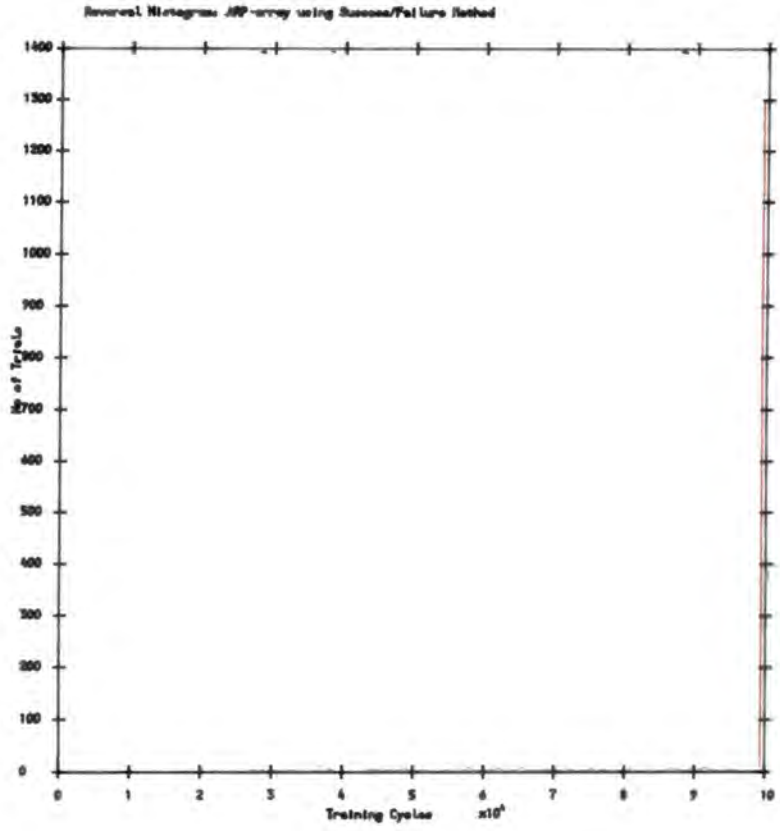


Figure 8.8a Reversal Histogram using A_{R-P} Success/Failure method

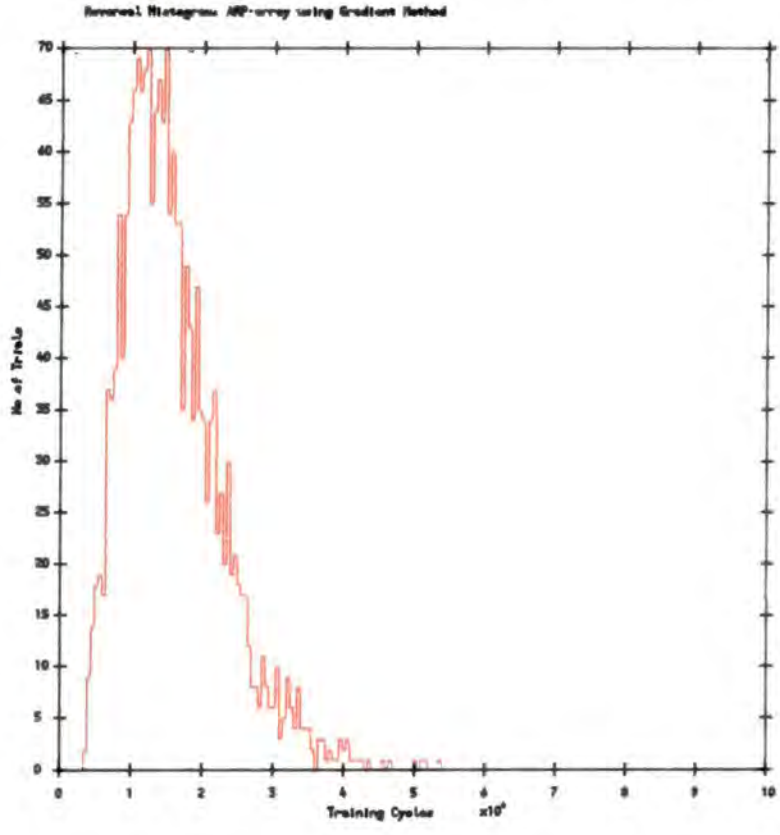


Figure 8.8b Reversal Histogram using A_{R-P} Gradient method

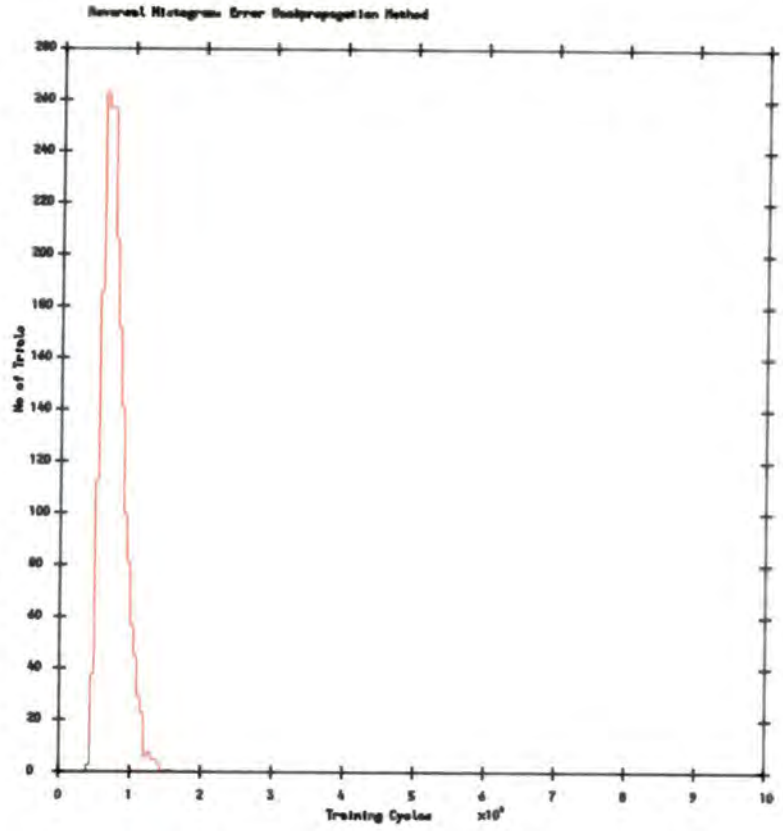


Figure 8.8c Reversal Histogram using error backpropagation

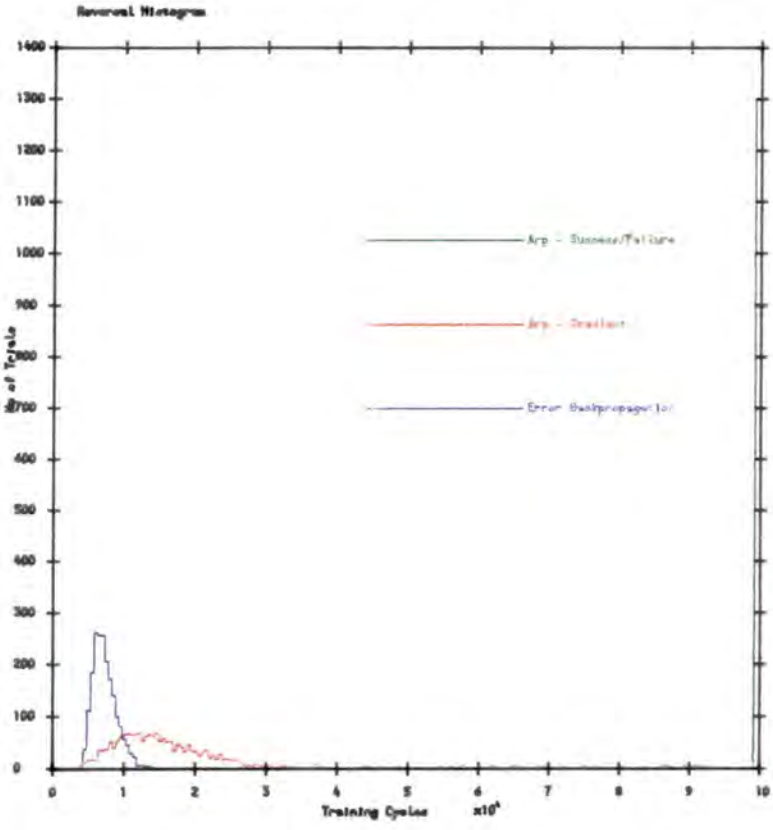


Figure 8.8d Reversal Histogram (all plots)

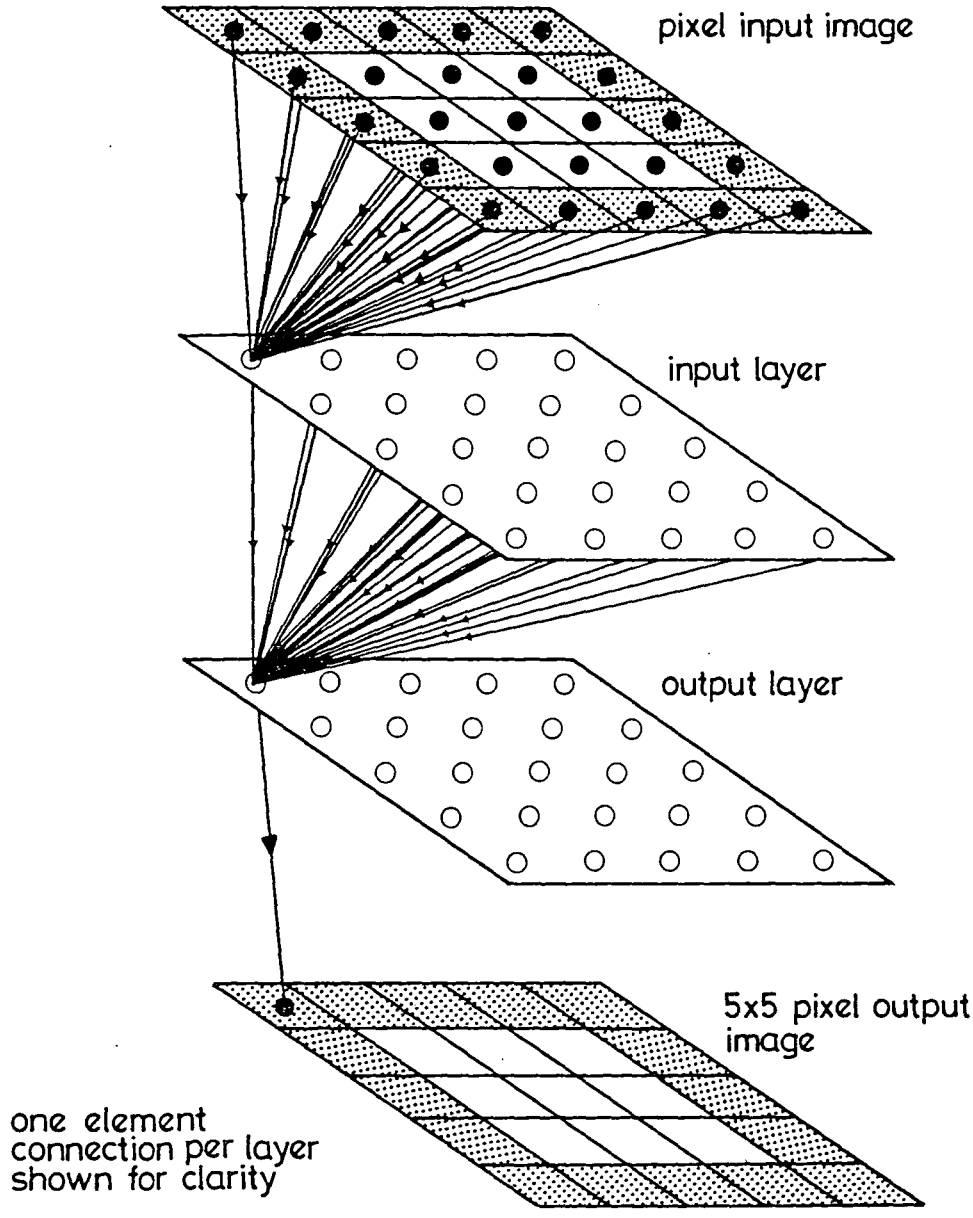


Figure 8.9 Zero Image topology

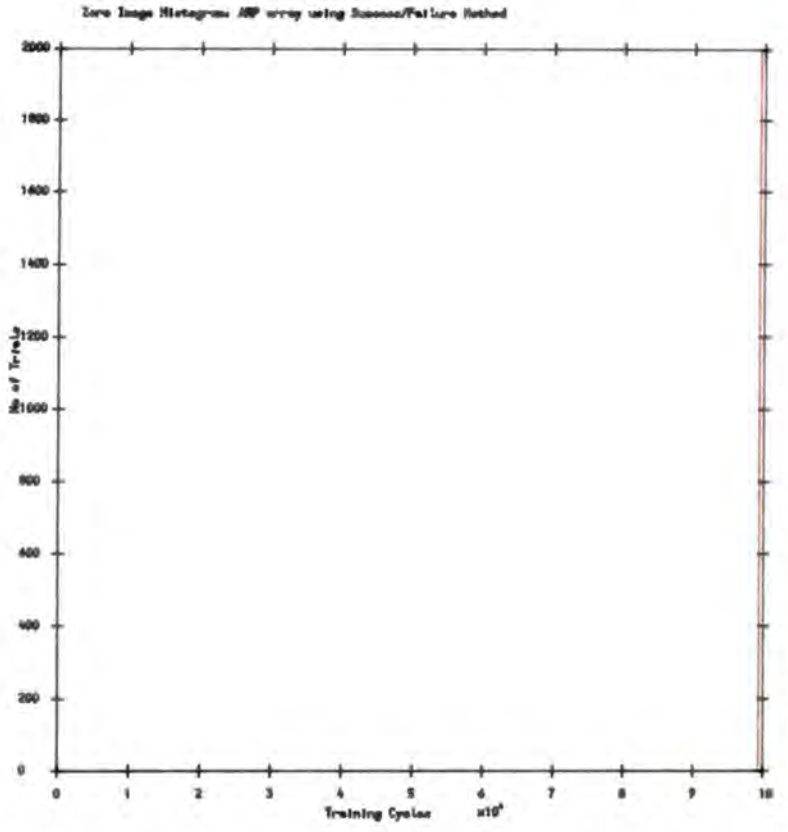


Figure 8.10a Zero Image Histogram using A_{R-P} Success/Failure method

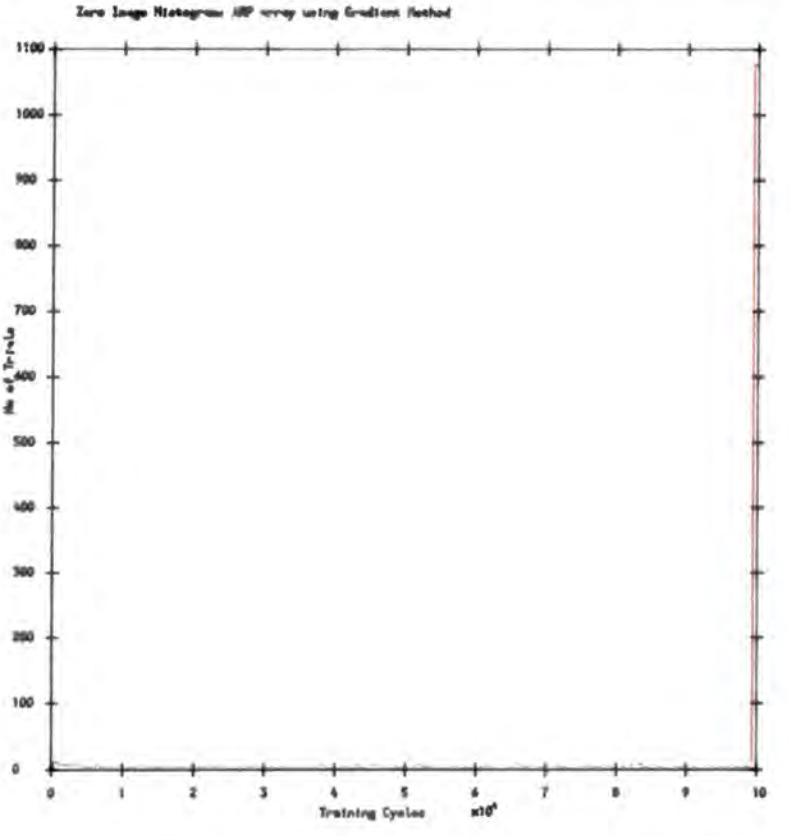


Figure 8.10b Zero Image Histogram using A_{R-P} Gradient method

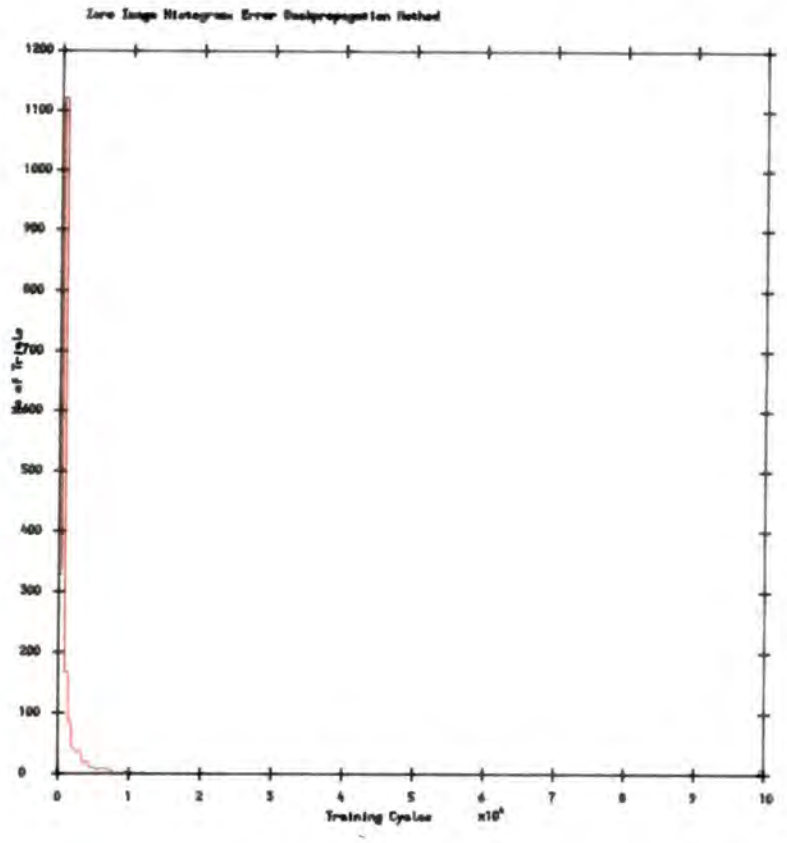


Figure 8.10c Zero Image Histogram using error backpropagation

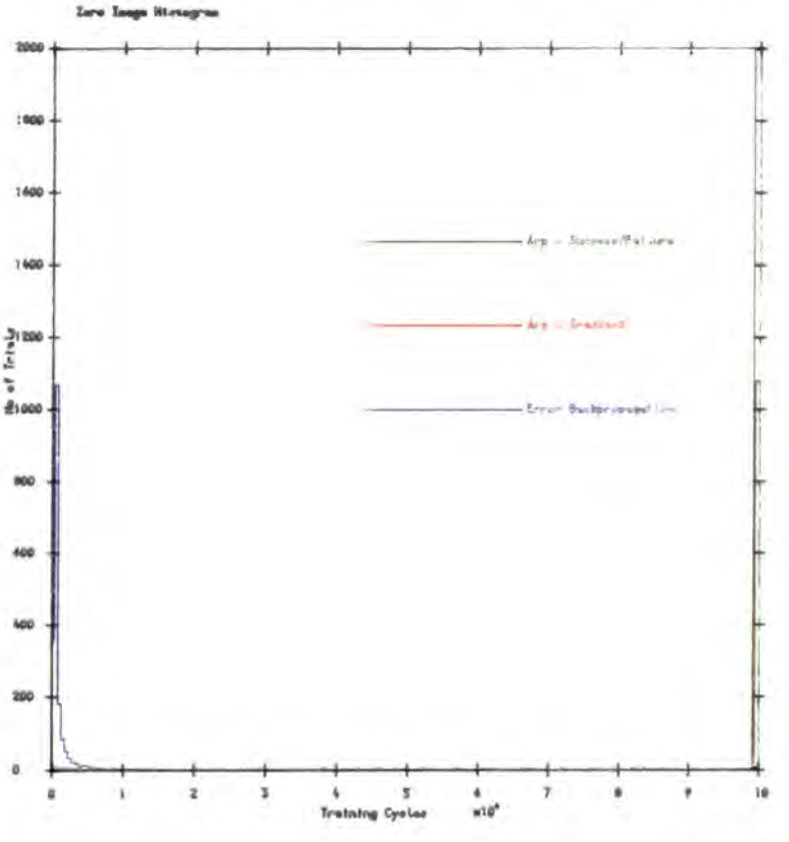


Figure 8.10d Zero Image Histogram (all plots)

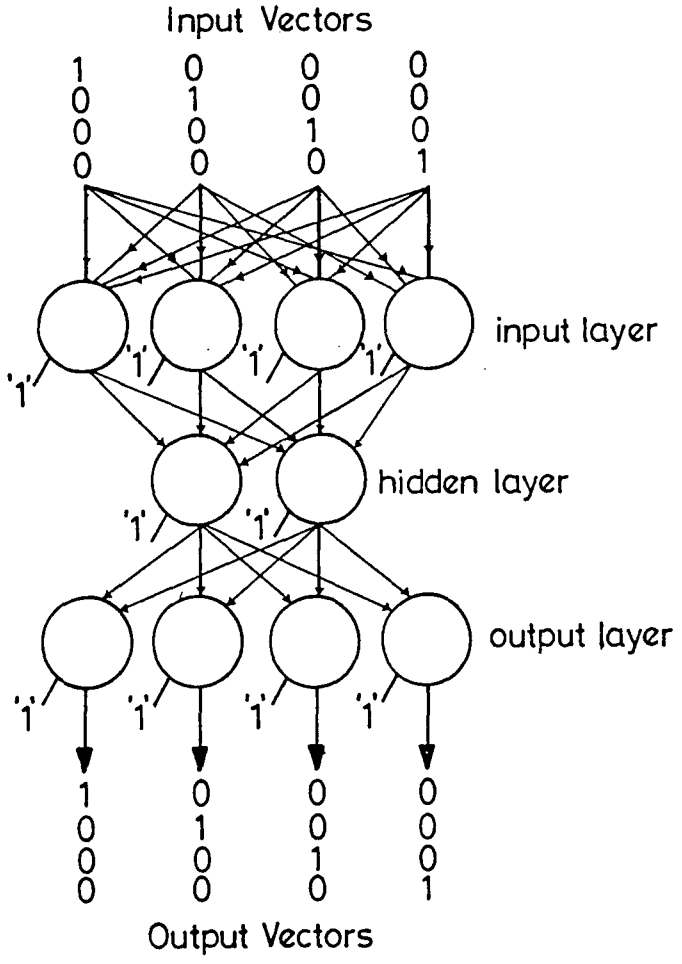


Figure 8.11 4–2–4 Channel Encoder topology

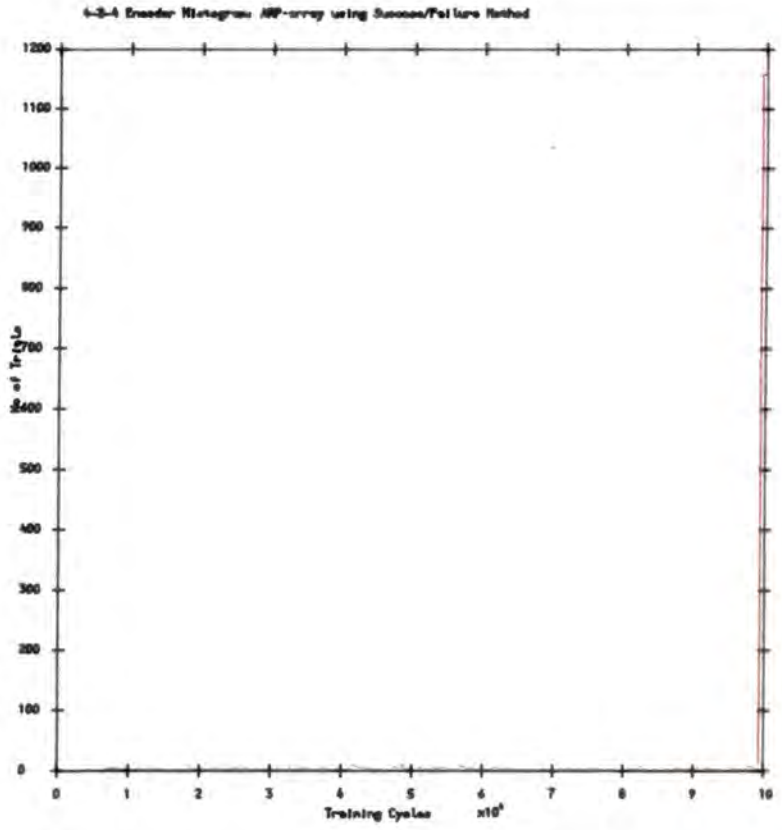


Figure 8.12a 4–2–4 Channel Encoder Histogram using A_{R-P} Success/Failure method

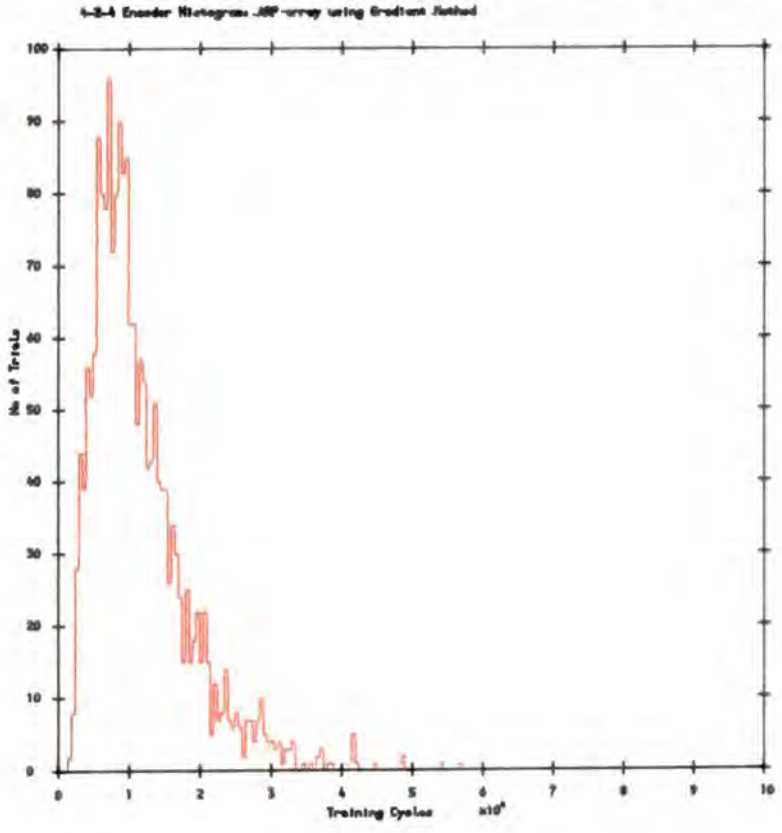


Figure 8.12b 4–2–4 Channel Encoder Histogram using A_{R-P} Gradient method

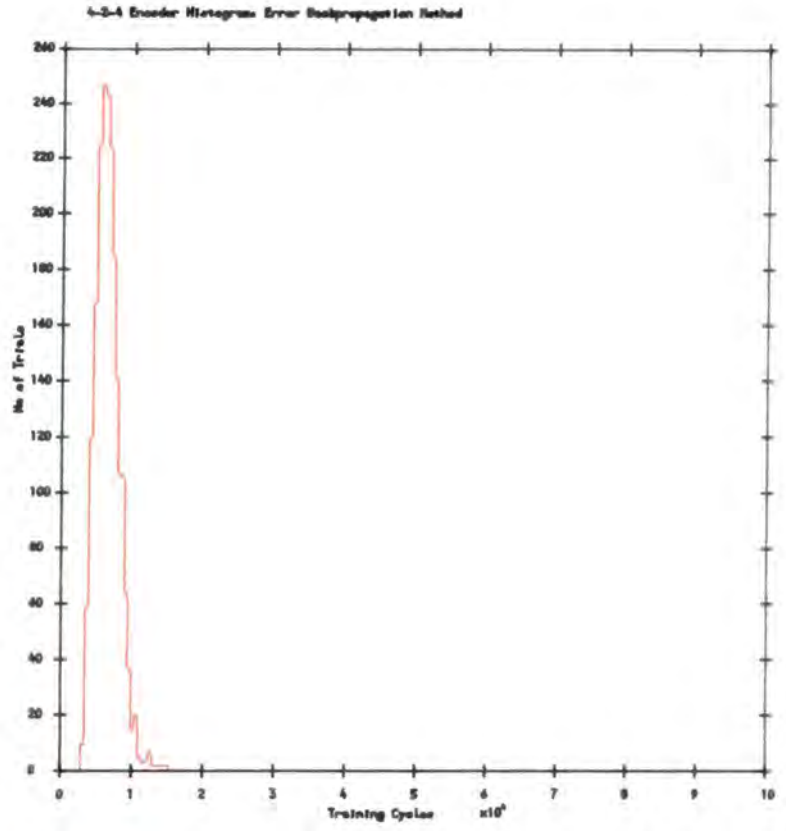


Figure 8.12c 4–2–4 Channel Encoder Histogram using error backpropagation

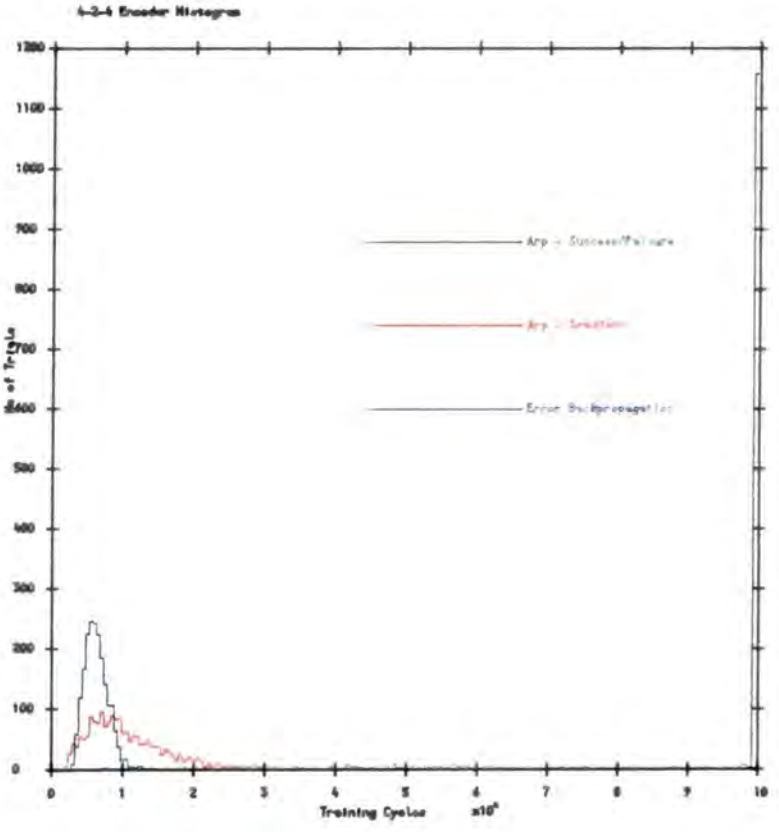


Figure 8.12d 4–2–4 Channel Encoder Histogram (all plots)

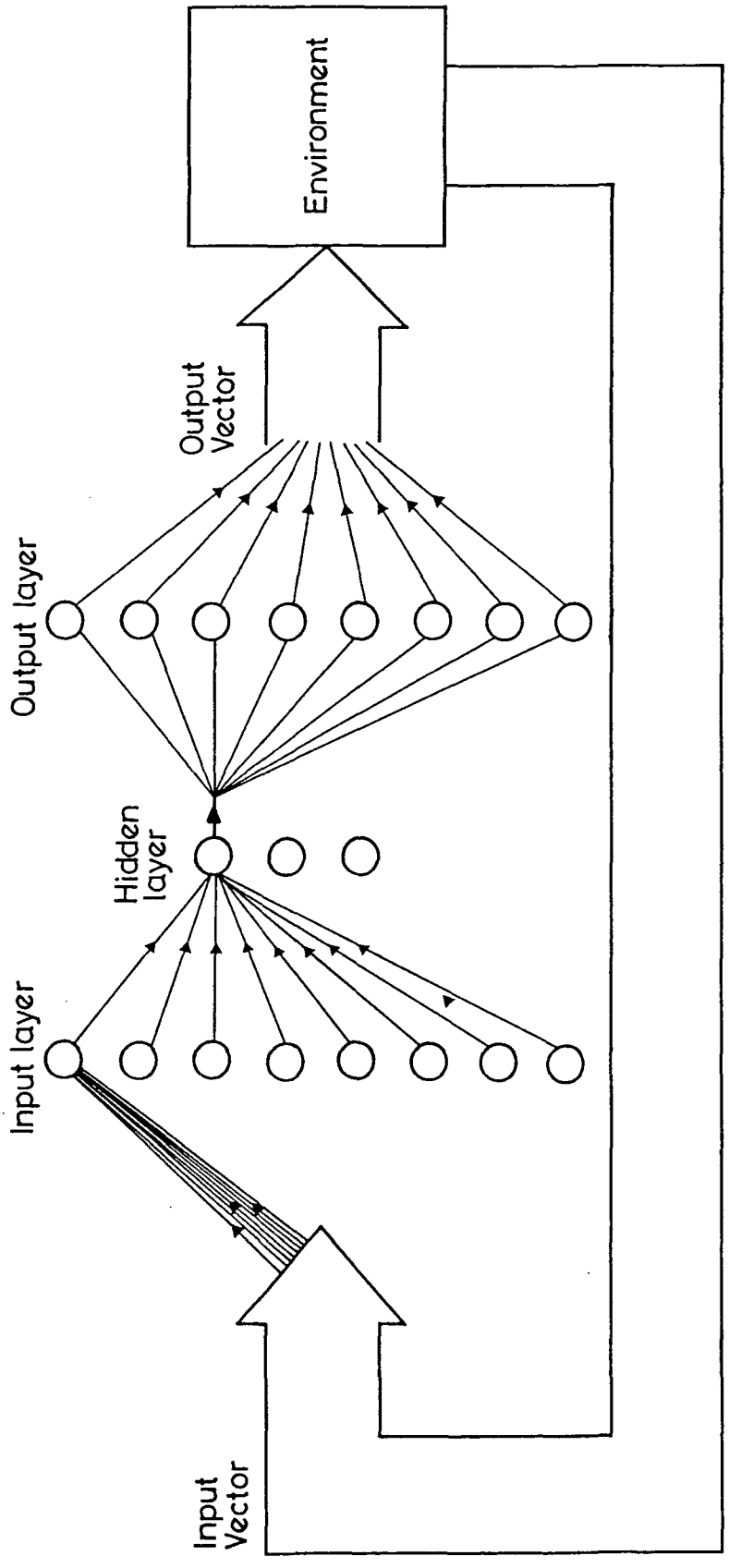


Figure 8.13 8–3–8 Channel Encoder topology

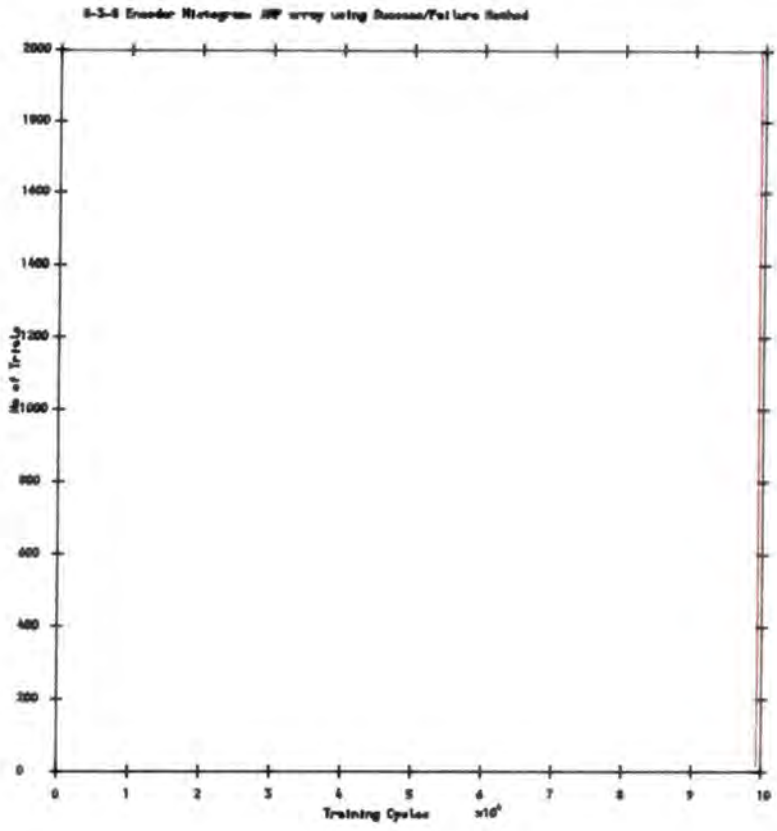


Figure 8.14a 8–3–8 Channel Encoder Histogram using A_{R-P} Success/Failure method

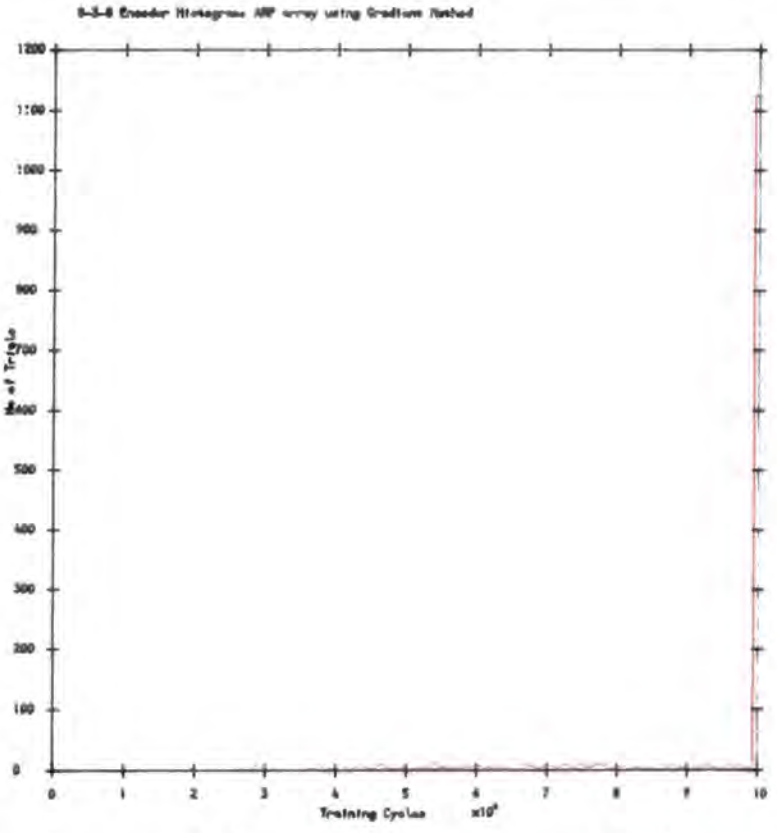


Figure 8.14b 8–3–8 Channel Encoder Histogram using A_{R-P} Gradient method

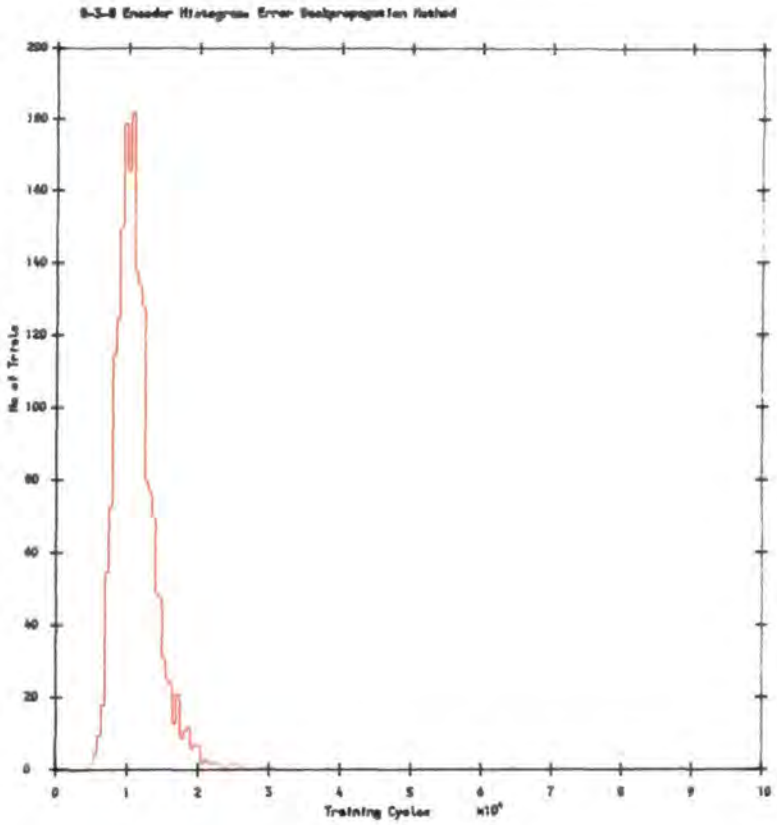


Figure 8.14c 8–3–8 Channel Encoder Histogram using error backpropagation

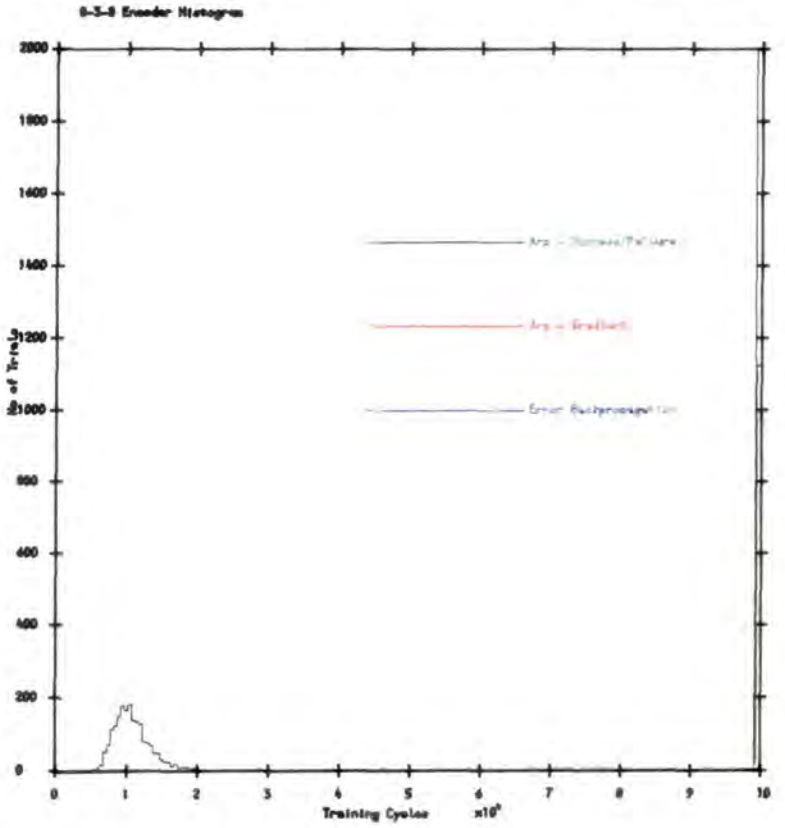


Figure 8.14d 8–3–8 Channel Encoder Histogram (all plots)

Table 8.1 - Exclusive OR truth table

Input	Output
0,0	0, 0
0,1	0, 1
1,0	0, 1
1,1	0, 0

Table 8.2 - Even Parity check truth table

Input	Output
0,0,0,0,0	0
0,0,0,0,1	1
0,0,0,1,0	1
0,0,0,1,1	0
0,0,1,0,0	1
0,0,1,0,1	0
0,0,1,1,0	0
0,0,1,1,1	1
0,1,0,0,0	1
0,1,0,0,1	0
0,1,0,1,0	0
0,1,0,1,1	1
0,1,1,0,0	0
0,1,1,0,1	1
0,1,1,1,0	1
0,1,1,1,1	0
1,0,0,0,0	1
1,0,0,0,1	0
1,0,0,1,0	0
1,0,0,1,1	1
1,0,1,0,0	0
1,0,1,0,1	1
1,0,1,1,0	1
1,0,1,1,1	0
1,1,0,0,0	0
1,1,0,0,1	1
1,1,0,1,0	1
1,1,0,1,1	0
1,1,1,0,0	1
1,1,1,0,1	0
1,1,1,1,0	0
1,1,1,1,1	1

Table 8.3 - Reversal truth table

Input	Output
0,0,0,0,0	0,0,0,0,0
0,0,0,0,1	1,0,0,0,0
0,0,0,1,0	0,1,0,0,0
0,0,0,1,1	1,1,0,0,0
0,0,1,0,0	0,0,1,0,0
0,0,1,0,1	1,0,1,0,0
0,0,1,1,0	0,1,1,0,0
0,0,1,1,1	1,1,1,0,0
0,1,0,0,0	0,0,0,1,0
0,1,0,0,1	1,0,0,1,0
0,1,0,1,0	0,1,0,1,0
0,1,0,1,1	1,1,0,1,0
0,1,1,0,0	0,0,1,1,0
0,1,1,0,1	1,0,1,1,0
0,1,1,1,0	0,1,1,1,0
0,1,1,1,1	1,1,1,1,0
1,0,0,0,0	0,0,0,0,1
1,0,0,0,1	1,0,0,0,1
1,0,0,1,0	0,1,0,0,1
1,0,0,1,1	1,1,0,0,1
1,0,1,0,0	0,0,1,0,1
1,0,1,0,1	1,0,1,0,1
1,0,1,1,0	0,1,1,0,1
1,0,1,1,1	1,1,1,0,1
1,1,0,0,0	0,0,0,1,1
1,1,0,0,1	1,0,0,1,1
1,1,0,1,0	0,1,0,1,1
1,1,0,1,1	1,1,0,1,1
1,1,1,0,0	0,0,1,1,1
1,1,1,0,1	1,0,1,1,1
1,1,1,1,0	0,1,1,1,1
1,1,1,1,1	1,1,1,1,1

Table 8.4 - 4–2–4 Channel Encoder truth table

Input	Output
0,0,0,1	0,0,0,1
0,0,1,0	0,0,1,0
0,1,0,0	0,1,0,0
1,0,0,0	1,0,0,0

Table 8.5 - 8–3–8 Channel Encoder truth table

Input	Output
0,0,0,0,0,0,0,1	0,0,0,0,0,0,0,1
0,0,0,0,0,0,1,0	0,0,0,0,0,0,1,0
0,0,0,0,0,1,0,0	0,0,0,0,0,1,0,0
0,0,0,0,1,0,0,0	0,0,0,0,1,0,0,0
0,0,0,1,0,0,0,0	0,0,0,1,0,0,0,0
0,0,1,0,0,0,0,0	0,0,1,0,0,0,0,0
0,1,0,0,0,0,0,0	0,1,0,0,0,0,0,0
1,0,0,0,0,0,0,0	1,0,0,0,0,0,0,0

Chapter Nine

Conclusions and Further Work

9.1 Stochastic Systolic Array

The first part of this thesis described the principles of stochastic encoding. Earlier research in this field concentrated mainly on control applications using stochastic learning automata, [11] and the thesis described how such techniques could be applied to stochastic digital signal processing using systolic, highly parallel architectures. The thesis showed how simple identical elements, performing the multiply/accumulate operation could be meshed into a novel array for digital signal processing.

This systolic array could perform the Discrete Fourier Transform and Cyclic Correlation operation, providing a good, time-dependent estimate of the true solution even with severe noise on the inputs. Using this approach, each processing element took part in the computation at all times, ie. no processor remained idle. Whilst the array could not necessarily compete with deterministic architectures in terms of noise-free performance, size and accuracy, the array showed fast convergence of the output average as the number of trials increased.

Further work in this area would have to include hardware implementations to remove the simulation time overhead. It would be interesting to observe the performance of the array in response to a changing input.

Direct input of stochastic quantities might well be possible using optical fibres and phototransistors, eliminating the input/output pin bottleneck with the attendant speed increase. The simplicity of the structure lends itself to the possibility of Wafer Scale implementation since fault tolerance could be built in. Perhaps implementation using optical logic elements might be possible at some time in the future, offering considerable speed advantages.

The main purpose of this primary research was to fully understand the limitations and explore the capabilities of stochastic processing for other deterministic problems with emphasis on neural networks. In particular, the research extended the study of the multiply/accumulate cell to a similar, multiple weighted input summing stochastic element in the learning arrays.

9.2 Neural Networks

The thesis explained the theory and application of stochastic learning automata which are probabilistic elements which learn to perform the 'best' actions in a changing, possibly noisy environment. The relatively new field of neural networks was then described, with two main classes of network, feed-forward such as Error Backpropagation, and recursive such as the Hopfield network and Boltzmann machine.

It was explained how the information in a neural network is distributed across the inter-element connections rather than localised. Consequently, minor damage to the network does not necessarily result in loss of information. Such encoding of information also endows the network with the capability to generalise from a partial training set. This may result in the capability to

reconstruct or recognise a degraded or partially obstructed image, depending on the task requirement.

It was further shown how the deterministic error backpropagation algorithm could be expressed using a simplified representation, as essentially two summing elements, one in the initial forward pass and the other in the error backpropagating reverse pass. From this description, it was shown how stochastic encoding principles could be applied, resulting in a stochastic error backpropagation algorithm based on a fundamental stochastic element.

This suggests the concept of a fundamental multiple weighted input summing stochastic element as the core component of an electronic neuron implementation. This then permits the use of a hybrid approach to the crucial weight updating algorithms. By such an approach, the possibility of using the best features from both global reinforcement learning algorithms and local error backpropagation methods becomes possible. Stochastic methods may also permit the possible use of optical approaches which could have particular importance for the massive cross-connection required at the inputs and outputs of each neuron, [104]. The thesis concluded with a number of evaluative simulations using a versatile experimental computer program.

9.3 Future Work - Hybrid Learning Mechanisms

The previous chapters of this thesis have been concerned with studies of reinforcement unsupervised learning elements such as A_{R-P} automata and the supervised learning methods of error backpropagation elements. These studies revealed two basic methods of implementing such networks; the first using

deterministic principles and the second, stochastic encoding. The stochastic approach, whilst time-consuming for software simulation is of considerable interest due to its parallel processing and hardware advantages for actual implementation. Furthermore, consideration of the global reinforcement signal which provides a feedback measure of A_{R-P} network performance as a whole revealed three potential coding methods, success/failure, gradient reward and virtual simulated annealing (VSA).

Future work should consider the merits of various hybrid approaches based on combinations of these methods as follows.

9.3.1 The Hybrid Element

Figure 9.1 shows the suggested form of the basic hybrid element. As before, the element forms the weighted summation of the inputs and computes a value based on the saturating output function. The element stochastically encodes this value as the probability of generating an output pulse, ie. firing.

The element has two feedback signals available to it. The first is a global reinforcement signal from the environment, as used by A_{R-P} automata, which offers information on how the network as a whole is performing. The coding of this reinforcement may be via success/failure, gradient or virtual simulated annealing (VSA) as explained in Chapter Six. The second, is local error information, backpropagated through the network as used by the stochastic error backpropagation method.

The element has the option of a number of different weight updates. The first is by using the reinforcement algorithm to compute a weight

update value based on the global reward signal. Note that the reinforcement algorithm requires that action and reaction must occur consecutively. It can also compute a second weight update value, using the error backpropagation method based on the local error at the element. Thirdly, the element, using this measure of local error, may use this as a local reinforcement signal and using the reinforcement algorithm, compute a weight update based on this signal.

In summary then, there are three potential weight update values which the element may compute for itself. The actual weight update chosen by the element may perhaps be the appropriate choice of average of some or all of these updates, or a scheme involving some alternating between choices.

9.4 Future Work - Networks

Aside from the hybrid element, there is much work that remains to be done on the actual networks themselves. The first is certainly the development of the appropriate systems theory which describes the learning process for the algorithms and enables prediction and optimisation of the networks to be made. The development of stochastic hardware demonstrators implementing the algorithms described in this thesis would be of great interest, particular for fast learning in large applications such as vision.

The networks primarily considered in this thesis have been of the feed-forward and recursive topologies. As explained earlier, these are only two possible examples out of the complete set of permutations of connections of neural elements. Other topologies may have equally interesting and different

properties.

Only spacial networks have been considered. It would be interesting to study temporal networks which provide specific outputs in sequence, in order to control some process in time. This might be implemented by a recursive network in the manner of a Finite State Machine, or by a set of feed–forward networks, each network output forming the input to the next, with each set of outputs forming the sequence of actions.

It is likely that only two dimensional structures could be considered for neural network implementation in VLSI, as opposed to the highly compact three dimensional structures of the biological neural nets. It would be interesting to see if research into novel devices, organic semiconductors and so on yields any properties which embody those required by an electronic neural element. A neural network array is certainly a very good candidate for Wafer Scale Integration (WSI) since the learning process should take account of faulty elements naturally during the arrays self–organisation.

Finally, it must be said that connectionist or neural architectures do not seek to replace earlier powerful symbolic Artificial Intelligence expert systems in any way. Rather, they complement the possibilities open to such a system by offering a set of primitive operations. e.g. 'reconstruct from exemplar set A' or 'is this recognised from exemplar set B'. In this way, the rule based system will examine the context of the situation to judge which set of weights (corresponding to a set of taught patterns) should be loaded into the network. Once trained, the speed of use of a neural network only depends on the propagation time through the array. During this time,

no silicon remains idle, all processors take part in the cumulative decision process.

9.5 Conclusions and Summary – Chapter Nine

In conclusion, stochastic arrays and learning networks have considerable potential. In the case of stochastic learning automata, there are major applications, for which the automata approach is clearly the best, however such an application for neural networks, whether global optimisation, or associative memory or something completely different, remains to be found.

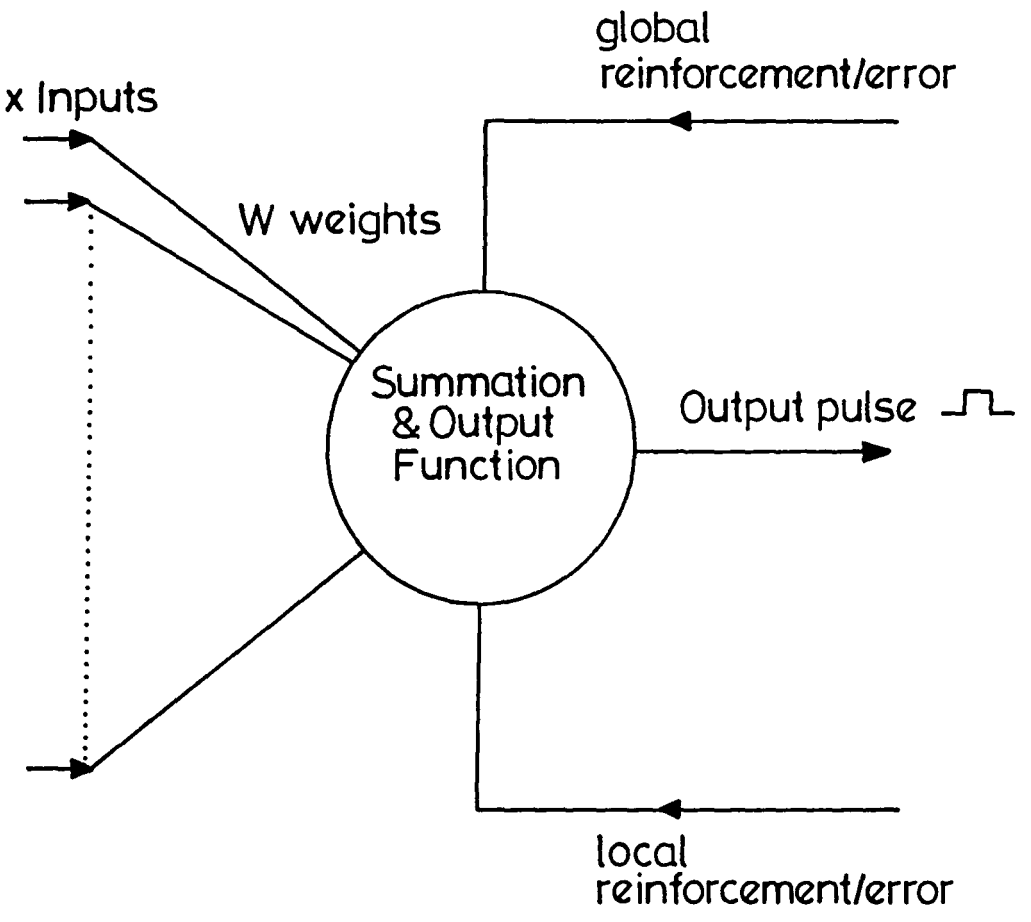


Figure 9.1 Hybrid Implementation of a Neural Element

References

1. Kohonen, T.:

   'Neural Networks',

   Vol 1, No. 1, 1988, pp. 3–16.

2. McClelland, J.L., Rumelhart, D.E., et al.:

   'Parallel Distributed Processing',

   Cambridge, MA., MIT Press, 1986.

   ISBN 0-262-18123-1

3. Jorgensen, C., and Matheus, C.:

   'Catching Knowledge in Neural Nets',

   AI Expert, 1986, Dec., pp. 30–38.

4. Ferry, G.:

   'Networks on the brain',

   New Scientist, 1987, July 16th., pp. 54–58.

5. Lippman, R.P.:

   'An Introduction to Computing with Neural Nets',

   IEEE ASSP, 1987, April, pp. 4–22.

6. Josin, G.:

   'Neural–Network Heuristics',

   Byte Magazine, 1987, Oct., pp. 183–192.

7. Jones, W.P., and Hoskins, J.:

   'Back–Propagation',

Byte Magazine, 1987, Oct., pp. 155–162.

8. Wasserman, P.D., and Schwartz, T.:
'Neural Networks',
IEEE Expert, pt. 2, pp. 10–15.

9. Kuffler, S.W., and Nicholls, J.G.:
'From Neuron to Brain',
Sinauer Associates, 1976, page 5.
ISBN 0-87893-441-3

10. McClelland, J.L., Rumelhart, D.E., et al.:
'Parallel Distributed Processing',
MIT Press, 1986, Volume 2, Chapters 20–21, pp. 333–389.
ISBN 0-262-18123-1

11. Mars. P., and Poppelbaum, W.J.:
'Stochastic and Deterministic Averaging Processors',
IEE Monograph, Peter Peregrinus Ltd, 1981.
ISBN 0-906048-44-3

12. Leaver, R.A., Mars, P., and Sarhadi, M.:
'Application of a Stochastic Systolic Array to Photon Correlation and LDA Techniques',
Proc. Second International Conf. on Laser Doppler Anemometry – Advances and Applications, Strathclyde, 1987.

13. Mars, P., and Narendra, K.S.:
'The use of Learning Automata in Telephone Communications Networks',

Automatica, 1983, Vol. 19, No. 5, pp. 495–502.

14. Mars, P., Narendra, K.S., and Chrystall, M.S.:
'Learning Automata Control of Computer Communications Networks',
Proc. of Third Yale Workshop on Applications of Adaptive Systems Theory, 1983, June, pp. 114–119.

15. Miller, A.J.:
'Digital Stochastic Computation',
PhD Thesis, 1976, University of Aberdeen.

16. Davenport, W.B.:
'Probability and Random Processes',
McGraw–Hill, 1970.
ISBN 07-015440-6

17. Rabiner, L.R., and Gold, B.:
'Theory and Application of Digital Signal Processing',
Prentice–Hall, 1975.
ISBN 0-13-914101-4

18. Rader, C.M.:
'Discrete Fourier Transforms when the number of samples is prime',
Proc. IEEE, Vol 56, No. 6, 1968, pp. 1107–1108.

19. Abbiss, J.B., Chubb, T.W., and Pike, E.R.:
'Laser Doppler Anemometry',
Optics and Laser Tech., 1974, December, pp. 249–261.

20. Narendra, K.S., and Thathachar, M.A.L.:
'Learning Automata - A Survey',

IEEE Trans., 1974, SMC-4, pp. 323–334.

21. Barto, A.G.:

'Learning by statistical cooperation of self-interested neuron-like computing elements',

Human Neurobiol., 1985, Vol. 4, pp. 229–256.

22. Barto, A.G., Anandan, P., and Anderson, C.W.:

'Cooperativity in Networks of Pattern Recognising Stochastic Learning Automata',

in 'Adaptive and Learning Systems', (ed. Narendra, K.S.),

Plenum Publishing Corporation, 1986, pp 235–246.

ISBN 0-306-42263-8

23. Poppelbaum, W.J., Afuso, C., and Esch, J.W.:

'Stochastic computing elements and systems',

AFIPS SJCC, 1967, Vol. 30, pp. 149–156.

24. Ribeiro, S.T.:

'Random-Pulse Machines',

IEEE Trans., 1967, EC-16, pp. 261–276.

25. Proc. of 'First Int. Symp on Stochastic Computing and its Applications',

Toulouse, France, 1978, Nov/Dec.

26. Gaines, B.R.:

'Stochastic Computing Systems',

in 'Advances in Information System Science', (ed. by Tou, J.T.),

Plenum Press, New York, 1969, pp. 37–172.

27. Miller, A.J., Brown, A.W., and Mars, P.:
'Adaptive logic circuits for digital stochastic computers',
Electron. Lett., 1973, Vol. 9, pp. 500–502.

28. Herault, J.:
'Nervous system considered as a stochastic processor: application to information processing',
Proc. First Int. Symp. on Stochastic Computing, Toulouse, France, 1978, paper 6.2, pp. 277–286.

29. Pollard, J.H.:
'A Handbook of Numerical and Statistical Techniques',
Cambridge University Press, 1977.
ISBN 0-521-29750-8

30. Miller, A.J., and Mars, P.:
'Theory and design of a digital stochastic computer random number generator',
Trans. IMACS, 1977, Vol. 19, pp. 198–216.

31. Miller, A.J., Brown, A.W., and Mars, P.:
'A simple technique for the generation of delayed maximum length binary sequences',
IEEE Trans., 1977, EC-26, pp. 808–811.

32. Miller, A.J., Brown, A.W., and Mars, P.:
'Moving-average output interface for digital stochastic computers',
Electron. Lett., 1974, Vol. 10, pp. 419–420.

33. Miller, A.J., Brown, A.W., and Mars, P.:
'Optimum criteria for output interfaces in digital stochastic computers',
Electron. Lett., 1975, Vol. 11, pp. 326–327.

34. Miller, A.J., Brown, A.W., and Mars, P.:
'Error reduction techniques in digital stochastic computers',
Trans. IMACS, 1977, Vol. 19, pp. 51–56.

35. Miller, A.J., Brown, A.W., and Mars, P.:
'Study of an output interface for digital stochastic computers',
Int. J. Elect., 1974, Vol. 37, pp. 637–655.

36. Miller, A.J., and Mars, P.:
'Optimal estimation of digital stochastic sequences',
Int. J. Sys. Sci., 1977, Vol. 9, pp. 683–696.

37. Mars. P., and Poppelbaum, W.J.:
'Stochastic and Deterministic Averaging Processors',
IEE Monograph, Peter Peregrinus Ltd, 1981, pp.7–13.
ISBN 0-906048-44-3

38. Kung, H.T., and Leiserson, C.E.:
'Systolic Arrays for VLSI',
in 'Introduction to VLSI Systems', (ed. Mead, C.A., and Conway, L.A.),
Addison Wesley, 1980.

39. Casasent, D., Jackson, J., and Neuman, C.:
'Frequency multiplexed and pipelined iterative optical systolic array

processors',

Applied Optics, 1983, Vol. 22, No. 1, pp. 115–124.

40. Jackson, J., and Casasent, D.:

'Optical systolic array processor using residue arithmetic',

Applied Optics, 1983, Vol. 22, No. 18, pp. 2817–2821.

41. Guilfoyle, P.S.:

'Systolic acousto-optic binary convolver',

Optical Engineering, 1984, Vol. 23, No. 1, pp. 20–25.

42. Hovanessian, S.A.:

'Introduction to Synthetic Array and Imaging Radars',

Artech House, 1980.

ISBN 0-89006-082-7

43. Arambepola, B., and Brooks S.R.:

'Algorithms and Architectures for Digital SAR Processing', (ed. Offen, R.J),

in 'VLSI Image Processing',

Collins, 1985.

ISBN 0-00-383055-1

44. Roberts, J.B.G., Darby, B.J., Herring, R.W., and McWhirter, J.G.:

'Radar Signal Processing Architectures and Algorithms in digital VLSI',

RSRE, Malvern, 1983, August, Tech. Report TR-5.

45. Abramowitz, M., and Stegun, I.S.:

'Handbook of Mathematical Functions',

Dover, New York, 1965, pp. 827–865.

46. Kung, H.T.:
'Why Systolic Architectures?',
Computer Magazine, 1982, Vol. 15, No. 1, pp. 37–46.

47. Kung, S.Y.:
'VLSI Array Processors',
IEEE ASSP, 1985, July, pp. 4–22.

48. 'Proc. Int. Workshop on Systolic Arrays',
(ed. Moore, W., McCabe, A., and Urquhart, R.),
IOP Publishing, 1987.
ISBN 0-85274-826-4

49. Kung, H.T., and Menzilcioglu, O.:
'Warp: A Programmable Systolic Array Processor',
SPIE, Vol. 495, pp. 130–136.

50. McClellan, J.H., and Rader, C.M.:
'Number Theory in Digital Signal Processing',
Prentice–Hall, 1979.
ISBN 0-13-627349-1

51. Ward, J.S.:
'Algorithms and Architectures for Digital Signal Processing',
PhD Thesis, Durham University, 1983.

52. Nussbaumer, H.J.:
'Fast Fourier Transforms & Convolution Algorithms',
Springer–Verlag, 1981.

ISBN 3-540-10159-4

53. McCanny, J.V., and McWhirter, J.G.:

'Implementation of Signal Processing Functions using 1-bit systolic arrays',

Electron. Lett., 1982, Vol. 18, pp. 241–243.

54. Ward, J.S., and Stanier, B.J.:

'Implementation of Convolution and Fourier Transforms using 1-bit systolic arrays',

Electron. Lett., 1982, Vol. 18, pp. 799–800.

55. McCanny, J.V., and McWhirter, J.G.:

'Bit level systolic array circuit for matrix x vector multiplication',

Proc. IEE., Vol. 130, No. 4, Aug. 1983, pp. 125–130.

56. McCanny, J.V., McWhirter, J.G., and Wood, K.:

'Optimised bit level systolic array for convolution',

Proc. IEE., 1984, Vol. 131, No. 6, pp. 632–637.

57. Thompson, C.D.:

'Fourier Transforms in VLSI',

IEEE. Trans. on Computers, Vol C-32, No. 11, 1983.

58. Gomez, S., Gonzalez, S., Hsu, D.D., and Kuo, A.E.:

'An application specific FFT processor',

Electronic Engineering, 1988, Vol. 60, No. 738, pp. 99–106.

59. Luikuo, G., and Magar, S.:

'A 500 MOPS DSP chip set',

ibid., pp. 109–113.

60. 'Special Volume on Learning Automata',

J. Cybern & Inf. Sci., 1977, Vol. 1, No. 2.

61. Tsetlin, M.L.:

'Automaton theory and modelling of biological systems',

Academic Press, New York, 1973.

62. Wilde, D.J.:

'Optimum seeking methods',

Prentice-Hall, 1964.

Library of Congress No. 63-20039

63. Mackie, N.J., and Mars, P.:

'Stochastic automata in non-stationary environments',

Proc. First Int. Symp. on Stochastic Computing, Toulouse, France, 1978, paper 7.3, pp. 321–344.

64. Mackie, N.J.:

'Theory and Application of Learning Automata',

PhD Thesis, 1980, Robert Gordon's Institute of Technology, Aberdeen.

65. Neville, R.G:, Nicol, C.R. and Mars, P.:

'Design of stochastic learning automata using adaptive digital logic elements',

Electron. Lett., 1978, Vol. 14, pp. 324–326.

66. Neville, R.G., Nicol, C.R. and Mars, P.:

'Synthesis of stochastic learning automata',

Electron. Lett., 1978, Vol. 14, pp. 206–208.

67. Neville, R.G., and Mars, P.:

'Hardware design for a hierarchical structure stochastic learning automaton',

J. of Cybern. & Inf. Sci., 1979, Vol. 2, No. 1, pp. 30–35.

68. Neville, R.G., and Mars, P.:

'Hardware synthesis of stochastic learning automata',

Proc. First Int. Symp. on Stochastic Computing, Toulouse, France, 1978, Paper 7.4, pp. 345–365.

69. Neville, R.G.:

'Synthesis of Stochastic Learning Automata',

PhD Thesis, 1980, Robert Gordon's Institute of Technology, Aberdeen.

70. Neville, R.G.:

'Synthesis of Stochastic Learning Automata',

ibid. Chapter Five.

71. Hopfield, J.J.:

'Neural networks and Physical Systems with Emergent Collective Computational Abilities',

Proc. National Academy of Science USA, 1982, Vol 79, pp. 2554–2558.

72. Hinton, G.E., Sejnowski, T.J., and Ackley, D.H.:

'Boltzmann Machines: Constraint satisfaction networks that learn',

Carnegie–Mellon University, Pittsburgh, PA., Tech. Report CMU-CS-84-119, 1984.

73. McClelland, J.L., Rumelhart, D.E., et al.:
'Parallel Distributed Processing',
MIT Press, 1986, Volume 1, pp. 282–317.
ISBN 0-262-18123-1

74. Rumelhart, D.E., Hinton, G.E, and Williams, R.J.:
'Learning representations by back–propagating errors',
Nature, 1986, Vol. 323, pp. 533–536.

75. Hopfield, J.J. and Tank, D.W.:
"'Neural" Computation of Decisions in Optimisation Problems',
Biol. Cybern., 1985, Vol. 52, pp. 141–152.

76. Sejnowski, T., and Rosenberg, C.R.:
'NETtalk: A Parallel Network that learns to read aloud',
John Hopkins University Technical Report, JHU/EECS-86/01, 1986

77. Sejnowski, T.J., and Rosenberg, C.R.:
'Parallel Networks that Learn to Pronounce English Text',
Complex Systems, 1987, Vol. 1, pp. 145–168.

78. Hinton, G.E., and Anderson J.:
'Parallel Models of Associative Memory',
Erlbaum, Hilsdale, N.J., 1981.
ISBN 0-89859-105-8

79. Fahlman, S.E., and Hinton, G.E.:
'Connectionist Architectures for Artificial Intelligence',
Computer, 1987, Jan., pp. 100–109.

80. Hinton, G.E.:

'Distributed Representations',

Carnegie–Mellon University, Pittsburgh, PA., Tech. Report

CMU-CS-84-157.

81. Feldman, J.A., and Ballard, D.H.:

'Connectionist Models and their Properties',

Cognitive Science, 1982, Vol. 6, pp.205–254.

82. Aleksander, I.:

'Advanced Digital Information Systems',

Prentice–Hall, 1984.

ISBN 0-13-0113050

83. Plonsey, R., Fleming, D.G.:

'Bioelectric Phenomena',

McGraw–Hill, 1969.

Library of Congress No. 69-17189-50342

84. Kuffler, S.W., and Nicholls, J.G.:

'From Neuron to Brain',

Sinauer Associates, 1976, pp. 12–13.

ISBN 0-87893-441-3

85. Eccles, J.C.:

'The Understanding of the Brain',

McGraw–Hill, 1977. ISBN 0-07-018865-3

86. Levy, W.B., Anderson, J.A., and Lehmkuhle, S.: (eds.)

'Synaptic Modification, Neuron Selectivity and Nervous System Or-

ganisation',

Lawrence Erlbaum Associates, 1985.

ISBN 0-89859-344-1

87. Hebb, D.O.:

'The Organisation of Behaviour',

Wiley & Sons, New York, 1949.

88. McClelland, J.L., Rumelhart, D.E., (eds).:

'Parallel Distributed Processing',

MIT Press, 1986, Volume 1, pp. 322–328.

ISBN 0-262-18123-1

89. McClelland, J.L., Rumelhart, D.E., (eds).:

'Parallel Distributed Processing',

ibid. pp. 365–422.

90. Stonham, T.J.:

'Practical Pattern Recognition',

in 'Advanced Digital Information Systems',(ed. Aleksander, I.),

Prentice–Hall, 1984, pp. 232–272.

ISBN 0-13-0113050

91. Rosenblatt, F.:

'Principles of Neurondynamics: Perceptrons and the Theory of Brain Mechanisms',

Spartan Books, Washington D.C., 1962.

92. Minsky, M.L., and Papert, S.:

'Perceptrons',

MIT Press , 1969.

ISBN 0-262-13043-2

93. Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P.:

'Optimisation by Simulated Annealing',

Science, 1983, Vol. 220, pp. 671–680.

94. Metropolis, N., Rosenbluth, M., Teller, A., and Teller, E.:

'Equations of state calculations for fast computing machines',

J.Chem.Phys., 1953, Vol. 21, pp. 1087–1092.

95. Eisberg, R.M.:

'Fundamentals of Modern Physics',

Wiley, 1961.

ISBN 0-471-23463-X

96. Hopfield, J.J.:

'Neurons with graded response have collective computational properties like those of two–state neurons',

Proc. National Academy of Science USA, 1984, Vol 81, pp. 3088–3092.

97. Hopfield, J.J., and Tank, D.W.:

'Computing with Neural Circuits: A Model',

Science, 1986, Vol. 233, pp. 625–633.

98. Grant, P.M., and Sage, J.P.:

'A comparison of neural network and matched filter processing for detecting lines in images',

Proc. AIP Conf 151, Neural Networks for Computing, Snowbird,

Utah, 1986.

99. Wallace, D.J.:

'Memory and Learning in a class of neural network models',

Proc. Workshop on Lattice Gauge Theory, Wuppertal, 1985, pp. 315–330.

100. Ackley, D.H., Hinton, G.E., and Sejnowski, T.J.:

'A learning algorithm for Boltzmann machines',

Cognitive Science, Vol. 9, 1985, pp. 147–169.

101. Szu, H.H., and Hartley, R.L.:

'Nonconvex Optimisation by Fast Simulated Annealing',

Proc. IEEE., 1987, Vol. 75, No. 11, pp. 1538–1540.

102. Graf, H.P., Jackel, L.D., and Hubbard, W.E.:

'VLSI Implementation of a Neural Network Model',

IEEE Computer, 1988, March, pp. 41–49.

103. Murray, A.F., Smith, A.V.W., and Butler, Z.F.:

'Bit Serial Neural Networks',

Proc. IEEE Conf. on Neural Information Processing Systems, Denver, 1987.

104. Abu-Mostafa, Y.S., and Psaltis, D.:

'Optical Neural Computers',

Scientific American, 1987, March, Vol. 256, pp. 88–95.

105. Widrow, G., Gupta, N.K., and Maitra, S.:

'Punish/Reward: Learning with a critic in adaptive threshold systems',

IEEE. Trans. Systems, Man and Cyber., 1973, Vol. 5, pp. 455–465.

106. Widrow, B., and Winter, R.:
'Neural Nets for Adaptive Filtering and Adaptive Pattern Recognition',
IEEE Computer, 1988, March, pp. 25–39.

107. Barto, A.G., and Anandan, P.:
'Pattern-Recognising Stochastic Learning Automata',
IEEE Trans. Syst., Man and Cyber., 1985, Vol. SMC-15, No. 3, pp. 360–375.

108. Barto, A.G., and Jordan, M.L.:
'Gradient Following without Back-Propagation in Layered Networks',
Proc. IEEE First Annual Conference on Neural Networks, San Diego, 1987, June.

109. Barto, A.G.:
'Game-Theoretic Cooperativity in Networks of Self-Interested Units',
in 'Neural Networks for Computing', (ed. Denker, J.S.),
Snowbird, Utah, AIP Conf. Proc. 151, 1986, pp. 41–46.

110. Williams, R.J.:
'Reinforcement Learning in Connectionist Networks: A Mathematical Analysis',
Institute for Cognitive Science, University of California, San Diego, ICS Tech. Report 8605.

111. Williams, R.J.:
'Reinforcement Learning Connectionist Systems',
College of Computer Science, Northeastern University, Boston, Tech.

Report NU-CCS-87-3, 1987, Feb.

112. Anderson, C.W.:

'Learning and Problem Solving with multilayer connectionist systems', Doctoral dissertation, University of Massachusetts, Amherst, COINS Technical Report 86–50, pp. 153-155.

Appendix One

Derivation of the Standard Deviation of a Bernoulli Sequence

A1.1 Introduction

As described in Chapter Two, stochastic encoding involves the use of Bernoulli sequences to convey information. It is convenient to describe the behaviour of a system operating under stochastic principles using the moments of the distribution. The first moment is the **expected** or **average** (mean) value. An estimate of the generating probability, p is obtained by taking the time average of the pulse sequence. As the length of the sequence tends to infinity, the estimate tends to the actual value. The estimate of the generating probability, $\widehat{P}$ obtained by taking the finite average as described will form a binomial distribution about the mean value $\bar{P}$. As the interval becomes infinite, the estimate becomes equal to the generating probability.

If one were to take independent estimates of $\widehat{P}$ from independent sequences of the same length, by taking a moving average at intervals over the sequence then the dispersion of each estimate about the mean of all the estimates may be expressed in terms of the second moment about the mean, the **variance** σ^2. It is convenient to take the positive square root of the variance, ie. the **standard deviation** σ as a measure which bounds the proportion of such estimates lying within the range

$$\bar{P} \pm \sigma \qquad (A1.1)$$

The Bernoulli distribution is only symmetric about the mean for a generating probability P=0.5. For this mid-range case only, 68% of the estimates taken as described should fall in the range $\bar{P} \pm \sigma$. In order to determine the standard deviation of a Bernoulli sequence, it is appropriate to introduce some standard definitions.

A1.2 Expectation

The Expectation of a probabilistic variable X is given by the expression:

$$E[X] = \sum_{x} x.P(x = X) \qquad (A1.2)$$

The sample mean or time average is given by;

$$\widehat{P} = \frac{1}{N}\sum_{i=1}^{N} x_i \qquad (A1.3)$$

The Expectation has important rules associated with it:

$$E[aX + b] = aE[X] + b \qquad (A1.4)$$

where X is the random variable and a and b are constants.

$$E[X + Y] = E[X] + E[Y] \qquad (A1.5)$$

where X and Y are both random variables.

A1.3 Variance

The variance, σ^2 is a measure of the dispersion of successive estimates about the mean or Expected value. For a sample sequence consisting of the

values x_1,x_2,x_3,....x_n, the variance of the sample, σ^2 is defined as;

$$\sigma^2 = \frac{1}{n-1}\sum_{j=1}^{n}(x_j - \bar{x})^2 \qquad (A1.6)$$

where $\bar{x}$ is the mean value of the sample. The variance of a random variable X may be defined in terms of its Expectation as:

$$VAR(X) = E[X^2] - E[X]^2 \qquad (A1.7)$$

The variance has the following rules which should be noted:

$$VAR(aX + bY) = a^2VAR(X) + b^2VAR(Y) + 2abCOV(X,Y) \qquad (A1.8)$$

where a and b are constants and X and Y are random variables. The function COV(X,Y) is known as the Covariance function of X and Y defined as:

$$COV(X,Y) = E[XY] - E[X]E[Y] \qquad (A1.9)$$

If X and Y are independent then:

$$E[XY] = E[X]E[Y] \qquad (A1.10)$$

and so:

$$COV(X,Y) = 0$$

A measure of Correlation between two random variables is CORR(X,Y) defined as:

$$CORR(X,Y) = \frac{COV(X,Y)}{\sqrt{VAR(X)VAR(Y)}} \qquad (A1.11)$$

Utilising these definitions, it is possible to derive an expression for the Standard Deviation of a Bernoulli sequence as follows.

A1.4 Analysis

Let A be a binary random variable which assumes the value 1 with probability P and consequently the value 0 with probability (1–P). ie:

$$P = P(A = 1)$$

and:

$$(1 - P) = P(A = 0) \qquad (A1.12)$$

The variance σ^2 of the estimate $\widehat{P}$ of the generating probability P is defined as:

$$\sigma^2 = VAR(\widehat{P}) \qquad (A1.13)$$

and therefore:

$$\sigma^2 = VAR(\frac{1}{N}\sum_{i=1}^{N} A_i) \qquad (A1.14)$$

where A_i is the value of A at the ith interval.

Using the property of variance:

$$VAR(aX) = a^2 VAR(X) \qquad (A1.15)$$

where a is a constant. Therefore (A1.14) becomes:

$$\sigma^2 = \frac{1}{N^2} VAR(\sum_{i=1}^{N} A_i) \qquad (A1.16)$$

and so:

$$\sigma^2 = \frac{1}{N^2} \sum_{i=1}^{N} VAR(A_i) \qquad (A1.17)$$

From (A1.7):

$$VAR(A_i) = E[{A_i}^2] - E[A_i]^2 \qquad (A1.18)$$

and so since A_i is a variable which can only take on the values of 0 or 1.

$$VAR(A_i) = E[A_i](1 - E[A_i]) \qquad (A1.19)$$

where:

$$E[A_i] = \sum_{j=1}^{N} A_j.P(A_i = A_j) \qquad (A1.20)$$

and for the binary case:

$$A_i = 1 \text{ with probability } P$$

and

$$A_i = 0 \text{ with probability } (1 - P)$$

Therefore:

$$\begin{aligned} E[A_i] &= 1.P + 0.(1 - P) \\ &= P \end{aligned} \qquad (A1.21)$$

Consequently:

$$VAR(A_i) = P.(1 - P) \qquad (A1.22)$$

Substituting this expression into (A1.17)

$$\sigma^2 = \frac{1}{N^2} \sum_{i=1}^{N} P.(1 - P) \qquad (A1.23)$$

and therefore:

$$\sigma^2 = \frac{1}{N^2} N.P(1 - P) \qquad (A1.24)$$

$$\sigma^2 = \frac{P.(1 - P)}{N} \qquad (A1.25)$$

The Standard Deviation, σ is therefore given by:

$$\sigma = \sqrt{\frac{P.(1 - P)}{N}} \qquad (A1.26)$$

It is important to consider the implications of this result. Clearly for P = 0 or 1, ie when the signal is deterministic, σ is zero. Maximum σ occurs when P = 0.5.

Appendix Two

Simulation programs – Stochastic Systolic Array

A2.1 Introduction

This Appendix briefly documents all programs (included in Volume 2), used to simulate the stochastic systolic array in order to perform complex correlation as described in Chapter Three. Additional programs are included which reorder the input and output data sequence so that the array may perform the Discrete Fourier Transform for an odd–prime length sequence. The program as documented is suitable for sequences up to 6 points in length. The listings contain information on array expansion for larger transforms.

A2.2 Stochastic Systolic Array

Program ARRAY.MAIN simulates an N x N array of stochastic cells in order to perform an N point cyclic correlation. The topology of the array is shown in Figure A2.1. The node numbering scheme shown is the one used by the program to connect cells together. The simulation performs complex correlation by simulating four arrays, one for each of the real–imaginary combinations. The program was designed to work for any desired correlation length (with the appropriate array declarations) although the local computer mainframe system memory limited simulations up to N=100 points in length. It was found in practice that a 47 point transform took over a whole weekend

of continuous terminal time to complete a 10,000 trial run and so this was the maximum size attempted. The basic Inner product cell is shown in Figure A2.2. This cell is described by subroutine CELL.SUB and forms the core of the simulation.

A2.3 Operation

The simulation expects input and reference data as shown by the examples INPUT and REFER included in Volume 2. These are in the format;

1. Title.
2. Number of points.
3. Real point 1, Imaginary point 1,
4. Real point 2, Imaginary point 2, etc.

The program is initialised by the use of subroutine INIT.SUB. This subroutine initialises the NAG random number generation routine for repeatable sequences and initialises all array nodes. Random numbers are then called using subroutine RAND.SUB.

Program I/O is handled by subroutine READER.SUB. This opens channel 1 for input data and channel 2 for reference data. Channel 3 opens an input file called SECOND. This was intended as a second input data sequence for brief experiments in non-stationary, ie. changing inputs and was not used for these simulations. The inputs are scaled after input to the program by the square root of the system dynamic range. The first output file opened is RESULT. This contains the final correlation output sequence in

the format of the input data files, on completion of a run. In other words, it contains the final estimate of the resultant correlation. There then follows a sequence of 6 output files, labelled as A1-A6. These files were used to record the simulation output at each trial for the N=6 simulations. This facility may be expanded for larger sequences (particularly noting the simulated shift register length). The program then works out the appropriate mapping arrangement for interconnection of the cells, for any N. The subroutine which performs this is MAPPER.SUB and uses the node notation of Figure A2.1. The program then requests the following information;

1. Window length - this is the width of the shift register over which the output averaging (moving average only) will be performed.

2. Averaging mode - if a '0' is returned, then a static average is performed over the entire sequence of trials. If a '1' is returned, then a moving average is performed over a window whose interval was previously input.

3. Standard Deviation of Noise (dB) - this is an input value which is defined as the ratio of the standard deviation of noise added to the input values to the maximum input range of the array ie.

$$\sigma(dB) = 20\log_{10}\{\frac{\sigma}{\sqrt{\text{OutputDynamicRange}}}\} \qquad (A2.1)$$

The dynamic range of the array, as explained in Chapter Three is simply the range of absolute values mapped on to the probability field 0 to 1. The program uses a NAG routine to add noise with the

Normal (Gaussian) probability distribution given by;

$$P(X) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[\frac{-(x-a)^2}{2\sigma^2}\right] \qquad (A2.2)$$

In this expression, a is the mean, and corresponds to the input value with zero noise, and σ is the standard deviation. An input value of -10000dB therefore corresponds to zero noise, whilst -3dB corresponds to addition of noise with a distribution whose variance (σ^2) is half the system dynamic range.

4. Maximum Time - this refers to the total number of trials to be performed.

5. Scale - this is the required output range of the array. Ideally, it should correspond to the maximum absolute value required at the output. For simulation of possible hardware counter mechanisms of limited capacity, it may be required that overscaling is performed to reduce the probability of coincident pulses during addition.

A2.4 Output

The subroutine OUTCNT.SUB performs either static or moving averaging depending on the selected input option. The routine AVERAG.SUB then combines the appropriate combinations of real and imaginary averaged outputs from the four virtual arrays and maps back from the probability domain to the appropriate output range and records these values in the appropriate files, as discussed earlier.

A2.5 Discrete Fourier Transform using Cyclic Correlation

As explained in Chapter Three, the Discrete Fourier Transform or DFT may be performed using a complex cyclic correlation by reordering the input and reference coefficients in a suitable way. The REFER.GEN program reads in an input sequence data file (containing the 47 point complex sample sequence representing a real pulse in the usual format) and outputs the appropriate reordered 46 point sequence. In similar fashion, INPUT.GEN generates the 46 point reordered 'twiddle' coefficients. It is here that some confusion may arise. The systolic correlator computes the cyclic correlation expression;

$$y_n = \sum_{k=0}^{N-1} h_k . x_{<n+k>} \qquad (A2.3)$$

where

$$n = 0, 1, \ldots, N\text{-}1$$

and

$$< > = \text{reduction MOD N}$$

Where the h coefficients are the reference (REFER) coefficients and the x values are the input data values (INPUT) which expands (for N=3) to ;

$$y_0 = h_0.x_0 + h_1.x_1 + h_2.x_2$$

$$y_1 = h_0.x_1 + h_1.x_2 + h_2.x_0 \qquad (A2.4)$$

$$y_2 = h_0.x_2 + h_1.x_0 + h_2.x_1$$

Now recall the 6 point correlation which computes the 7 point DFT from

Chapter Three;

$$\begin{bmatrix} X'_1 \\ X'_3 \\ X'_2 \\ X'_6 \\ X'_4 \\ X'_5 \end{bmatrix} = \begin{bmatrix} W^1 & W^3 & W^2 & W^6 & W^4 & W^5 \\ W^3 & W^2 & W^6 & W^4 & W^5 & W^1 \\ W^2 & W^6 & W^4 & W^5 & W^1 & W^3 \\ W^6 & W^4 & W^5 & W^1 & W^3 & W^2 \\ W^4 & W^5 & W^1 & W^3 & W^2 & W^6 \\ W^5 & W^1 & W^3 & W^2 & W^6 & W^4 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_3 \\ x_2 \\ x_6 \\ x_4 \\ x_5 \end{bmatrix} \qquad (A2.5)$$

and $W^7 = 1$.

It may be seen upon comparing equations A2.4 and A2.5 in terms of structure, that the x input values for the DFT case are treated as **reference** values and the h 'twiddle' coefficients are the **input** values. This is because the method correlates the reordered sine and cosine coefficients with the reference input sequence. The resulting correlation output from the array can then be input to the program DFT.REARRANG, in the usual format which will reorder the sequence and insert the appropriate DC value as the first term and output the resulting 47 point sequence corresponding to the DFT of the original 47 point sequence.

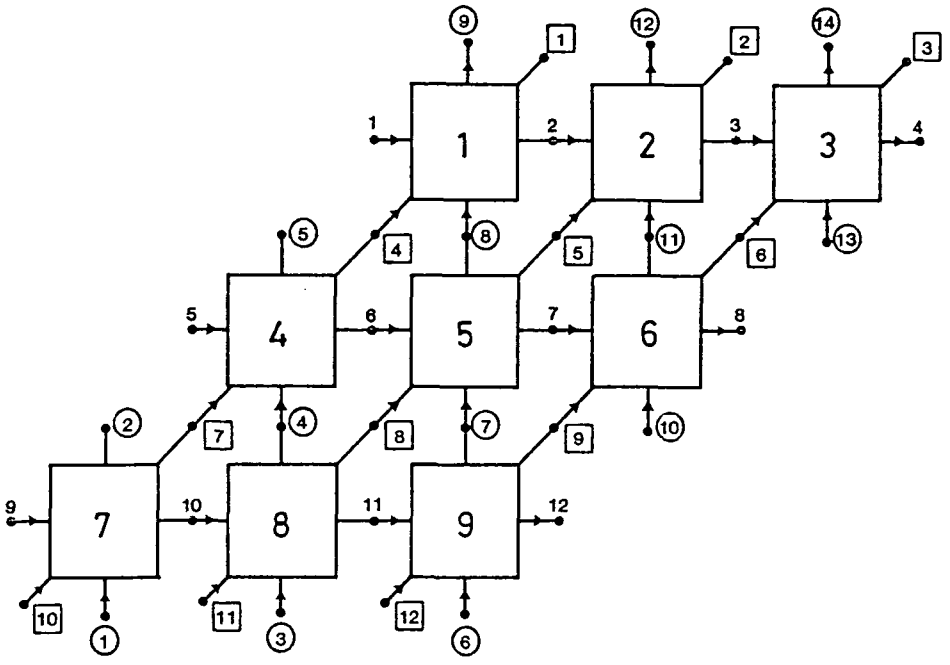


Figure A2.1 Systolic array – nodal topology

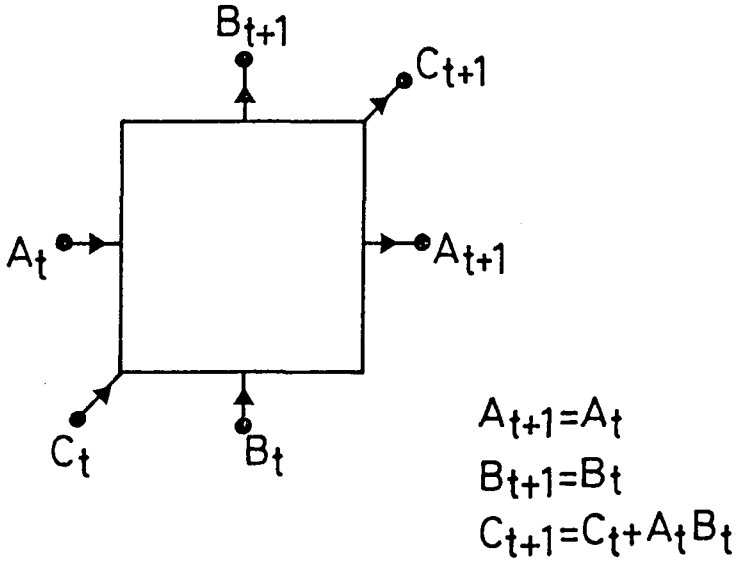


Figure A2.2 Inner Product Cell

Appendix Three

VLSI Implementation – Stochastic Systolic Array

A3.1 Introduction

During the early work on the stochastic systolic array, as described in Chapter Three, it was proposed that since the processor consisted of simple replicated cells, it would be an ideal subject for a Very Large Scale Integration (VLSI) design exercise. This appendix describes this work, undertaken by postgraduate students Stuart Taylor and John Hartmann as part of the requirements for the Durham Master of Engineering degree and has been included for completeness. Although the author is particularly grateful to his two colleagues for the many interesting discussions on the design practicalities of such a device, all credit for the the work reported here is theirs.

A3.2 Initial Functional Description of the Chip

The original design for the chip considered a 6 x 6 cell array. It was intended for the design to be expandable, such that it would be possible to link similar chips together into larger arrays. In addition to the stochastic output from the chip, a moving average circuit was also included which could selectively provide an estimate of the generating probability for any of the input or output stochastic lines. The design of the chip was split into two halves. The design of the cells and I/O was carried out by Stuart Taylor and

the averaging circuit was designed by John Hartmann. Figure A3.1 shows the initial overall array design and Figure A3.2 the initial block diagram. The input, output, carried data (from other chips) and reference stochastic data were connected to the chip along 6 single bit lines. Positive and negative (DLSR representation) values were accomplished by time-multiplexing these lines, effectively giving 12 input lines to each but doubling the time necessary to input quantities. The time division multiplexing allowed values on the positive line to be processed in one phase of the clock and the values on the negative line to be processed on the second phase of the clock. The averaged output data values were mapped onto the range -255 and +255, represented in signed binary format. These were output via the 9 parallel pins. The input data was latched using master-slave combination latches. Due to the time division multiplexing, the main array structure operated at half the input/output frequency.

A3.3 Input/Output Pin Details

The design was limited to 40 pin packages, although commercial packages can be considerably more. The input data fields account for 6 x 3 pins and the output data field and averaged field require 6 + 9. The supply and control requirements were, one pin for each of the following; v_{dd}, v_{ss}, clock phase 1, clock phase 2, select (for averaging) and reset. This made a total of 6 pins and a grand total of 39 pins, leaving one spare.

A3.4 Conclusions

During the design process, it was decided to revise the design to a 3 x 3 array due to both insufficient time and insufficient silicon space problems. This removed the pin out problem, together with the clock generator, input demultiplexor and output multiplexor. The addition correction mechanism used in each cell had restricted capacity (± 3) to save space and was implemented using a small Finite State machine. The chip was fabricated in 3μ CMOS. It was found that although the cell design was a non-trivial design, the fact that once it was designed, it could be simply replicated to form the array was very much appreciated, particularly in view of the local connections.

As it was a student project, the array had to share silicon space with other projects on the chip. The full chip is shown in Figure A3.3, and the actual array is shown in the magnified area of Figure A3.4. The 3 x 3 cells are clearly visible. Stuart Taylor performed planned array tests which were successfully compared against predicted HILO simulations indicating that the chip was operational although no time remained for more detailed analysis/breadboarding. All in all, the design of the array proved a very worthwhile exercise. Although the facilities available restricted the space and packaging necessary to fabricate a large array, the basic design was shown to be feasible and a commercial design would be much more compact, containing more cells and connections, certainly as many as 31 x 31 cells per chip.

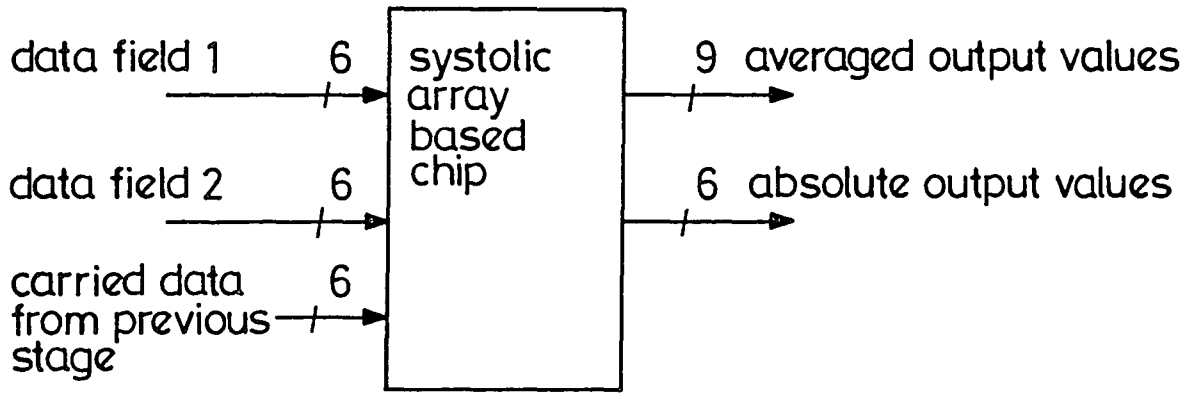
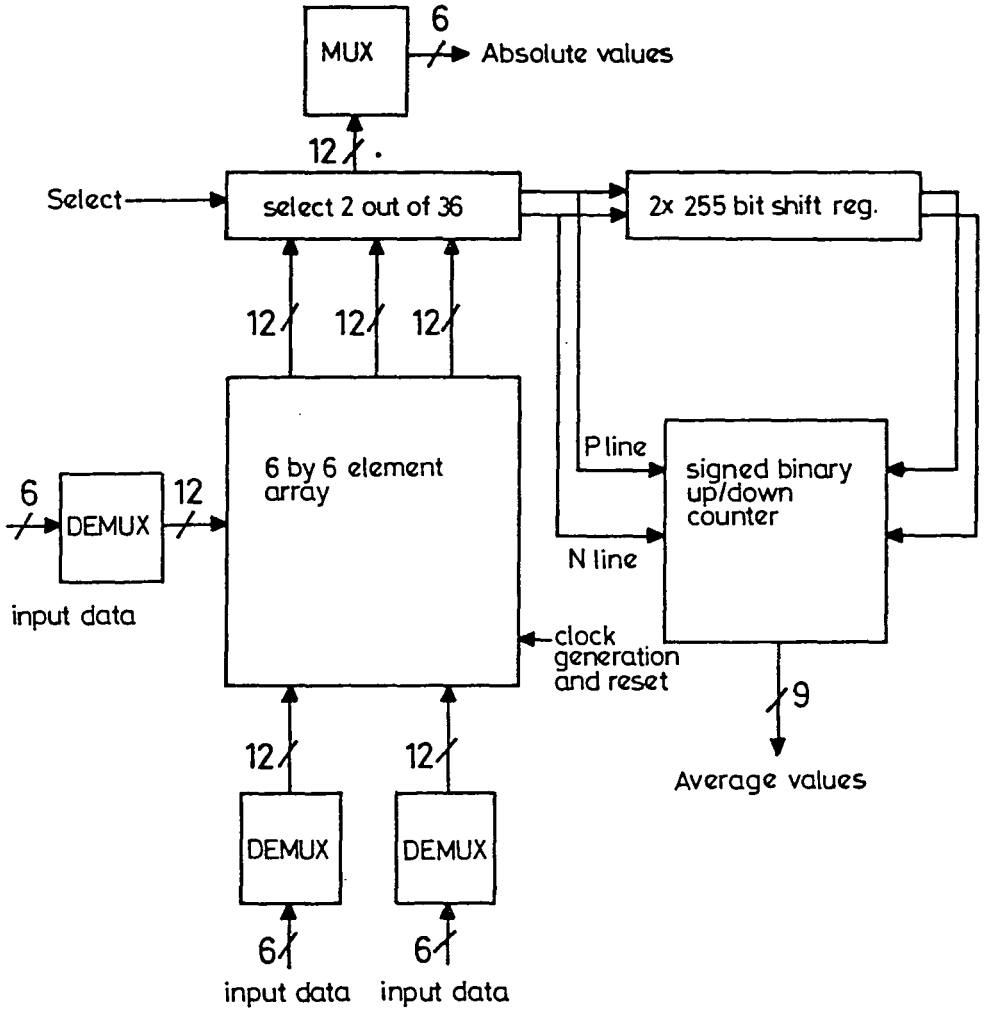


Figure A3.1 Basic array overview



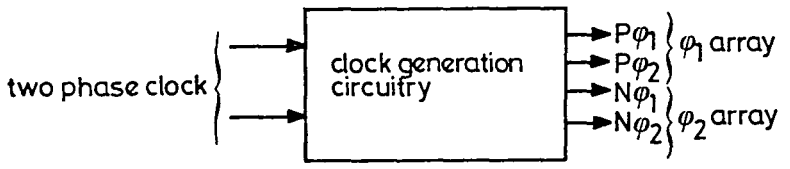


Figure A3.2 Chip block diagram

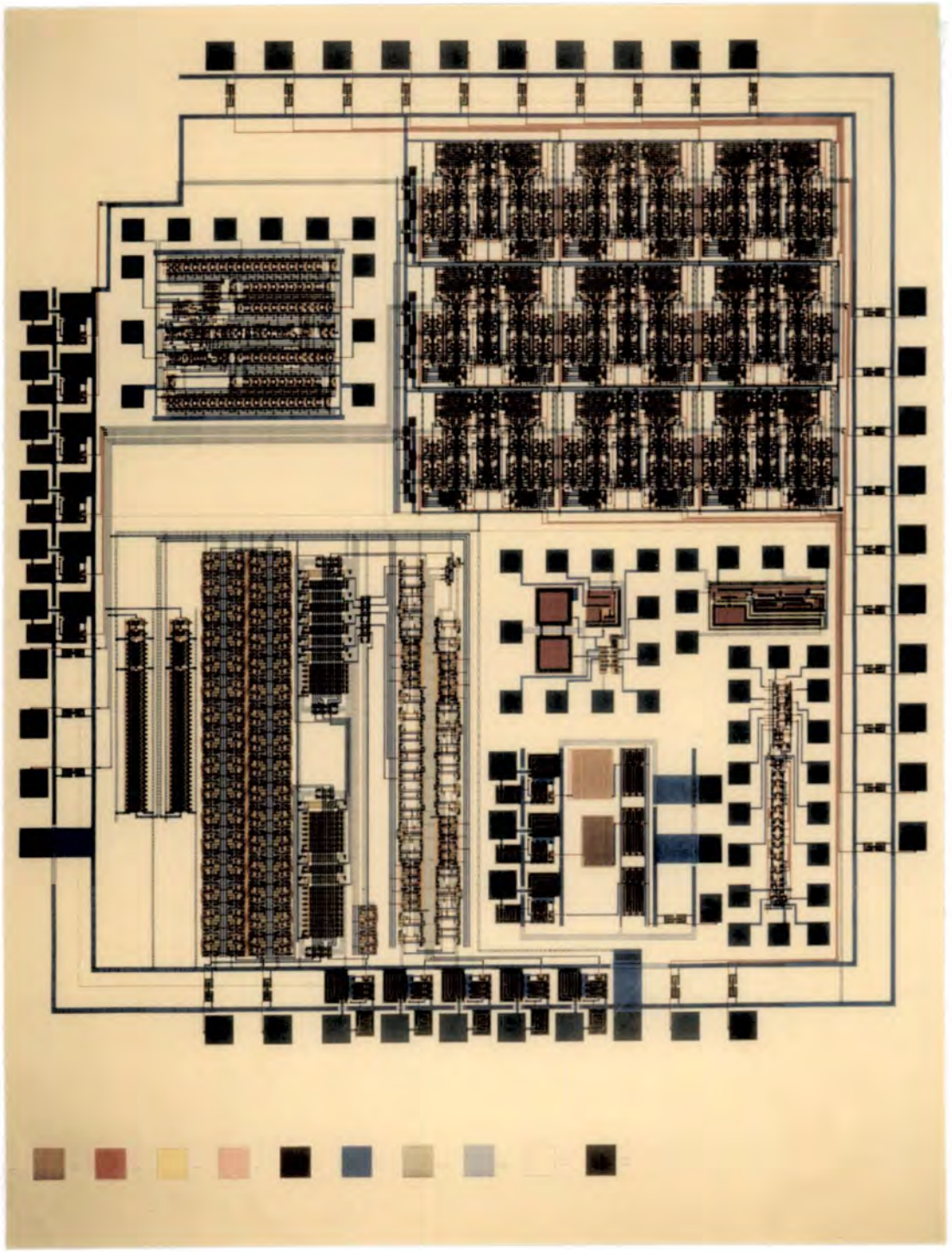


Figure A3.3 Completed chip design

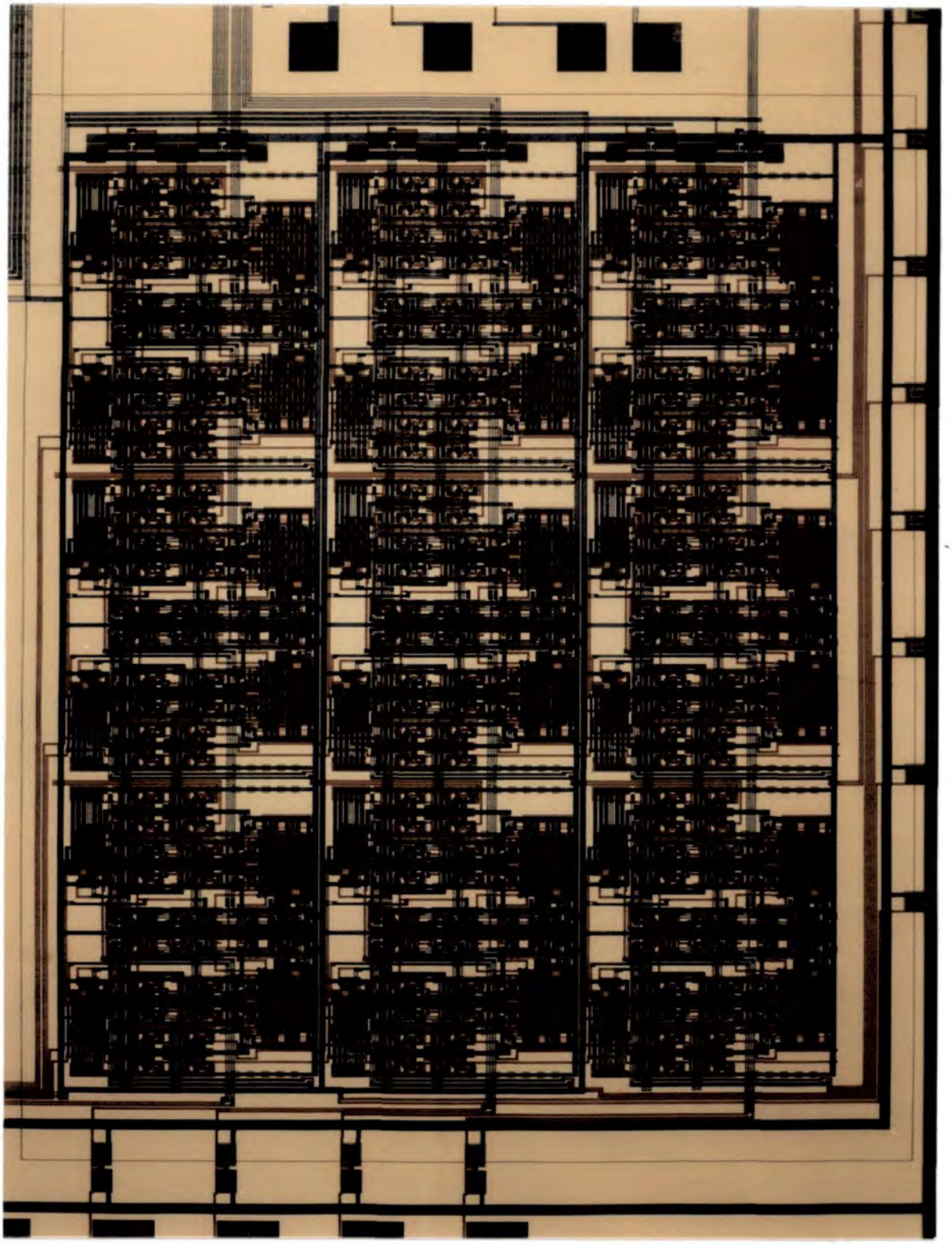


Figure A3.4 Completed array design

Appendix Four

Neural Network Simulation Program

A4.1 Introduction

This appendix describes the major neural network simulation program included in Volume 2 and used for all multilevel simulations and performance evaluations. The program was developed gradually over the period of the research and contains many features for both configuration and graphical output. Since the program was large, complex and took up to 2 minutes to compile, it was considered desirable to control all variations in input–output vector and parameter information using data files, input at the start of a run together with a series of interactive questions from the program to control configuration and output requirements. Six common modes of operation were identified and these were introduced as default settings enabling fast configuration.

A4.2 Input–Output vectors

The program was designed to accept input vectors of dimension X x Y requiring an input layer of X x Y elements. The output vector dimension and therefore output layer consists also of X x Y elements. The number of layers in the model, ie. depth was selectable from one to five layers with each layer consisting of X x Y elements. Therefore the whole array can be considered

as a network of X x Y x Z elements. Although many of the networks in the literature have many more 'hidden' elements than 'visible' elements, it was considered that the method adopted was more suitable for the problems it was required to examine. An example of the input-output data file EXOR shows the method used for problem entry. Some variables were input this way during early development and then discarded due to alternative methods. The format for consistency was retained and these variables are denoted by (Dummy) indicating their disuse. EXOR is prepared using the FORTRAN77 program ARP.PREP and consists of the following;

1. File Title.
2. Image Length.
3. Image Width.
4. Image Depth (Dummy).
5. Number of Input-Output vector combinations.
6. Weight file title (Dummy).
7. Input Vectors (in order).
8. Output Vectors (in order).
9. Corrupted Input Vectors (in order).

The input vectors were input for the second time in a corrupted form in order to evaluate the possible use of the array for reconstruction of a corrupted input to the nearest previously trained case.

A4.3 Parameters

Several other parameters were included in a general parameter file PARAM. These were, in order;

1. λ - A constant which was used to describe the A_{R-P} learning rate and also used for the error backpropagation weight update constant.
2. Array Length.
3. Array Width.
4. Array Depth.
5. ρ - A_{R-P} weight update constant, also used as the error backpropagation 'acceleration' term.
6. Temperature - used to modify the gradient of the element response curve.
7. Reward probability.
8. Punish probability.
9. Maximum number of learning trials in a cycle.
10. Maximum number of post-training trials.
11. Maximum number of cycles.
12. Proportion of output clamped over input in feedback mode.
13. Percentage correct learning criterion.
14. Number of independent re-runs of program.
15. Degree of batching.
16. Length of shift register for stochastic averaging.

A4.4 Program settings

On running the program, the user is prompted for the name of the data file containing the Input–Output vector information, e.g. EXOR. The program then asks whether the user requires one of the six default settings incorporated, these are;

1. A_{R-P} neural network using success/failure global reinforcement.
2. A_{R-P} neural network using gradient method reinforcement.
3. A_{R-P} neural network using local reinforcement (suitable for single level array only).
4. Deterministic error backpropagation.
5. Hopfield model (suitable for single level array only).
6. Stochastic error backpropagation.

If the program defaults are not selected then the following information is requested (the functions of the non–obvious ones are described in the next section).

1. Is a connection plot required? if so, which of the two formats is required – either the block element plot or the line element–linking plot.
2. Is an element weight plot required? if so, which element (note that for reasons of memory, only trial lengths of up to 10000 may be used with this option).
3. Is a global or local reinforcement scheme required (specify local for error backpropagation).
4. If global is requested, is this direct (reciprocal) or indirect (success

or failure).

5. Clipped or Unclipped training noise.

6. Clipped or Unclipped exercising noise.

7. Clipped or Unclipped gaussian noise.

8. Is it required to observe the noisy input image response? if so, which image.

9. Does the weight file contain previous weights? if so, is it just required to exercise the network.

10. If the number of input–output vectors is 1, is it required to intersperse these with random inputs? if so, should the network have a null (rather than random) response.

11. Is it required to plot the final output vector.

12. Is it required to plot the reinforcement levels.

13. Is it required to introduce selective feedback.

14. Is it required to use the error backpropagation error function (as opposed to the checksum function).

15. Is it required to impose symmetry on the weights.

16. Is a Hopfield–style test required.

17. Is a gradient reinforcement method required.

18. Is the stimulus input required.

19. Is it required to zero the input to an element corresponding to its own coordinates.

20. Is it required to plot the input/output training vectors.

21. Is a Hopfield, ARP or Error Backpropagation element required.

22. Is it required to randomise the weight matrix first.

23. Should a stochastic method be used.

When the program is run, the settings and inputs are automatically displayed. An example run, for the Exclusive OR problem, using an A_{R-P} success/failure reinforcement network (Default setting 1) case is given in Volume 2.

A4.5 Notes on Program settings

There were a number of interesting features built into the program to enable brief study of feedback and noise performance, but not as a major part of the research. These were;

1. Clipped or Unclipped Training Noise: The facility for using random inputs during training was incorporated. The random number was generated using a uniform distribution, providing values between or equal to 0 and 1. These could be directly input (Unclipped) or rounded to either 0 or 1 (Clipped).

2. Clipped or Unclipped Exercising Noise: The facility for using random inputs during exercising the array, after training was incorporated. The random number was generated using a uniform distribution, providing values between or equal to 0 and 1. Again, these could be directly input (Unclipped) or rounded to the nearest value, 0 or 1 (Clipped).

3. Clipped or Unclipped Gaussian Noise - Noisy image response: It was decided to examine the use of a trained array to reconstruct an image, using the Hopfield style feedback mode described later in this section.

The input image was corrupted by taking the original binary image values as the mean of a Gaussian Distribution. The program used a NAG routine to add noise with the Normal (Gaussian) probability distribution given by;

$$P(X) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[\frac{-(x-a)^2}{2\sigma^2}\right] \qquad (A4.1)$$

In this expression, a is the mean, and corresponds to the input value with zero noise, and σ is the standard deviation. The program evaluates the percentage of trials correct, plotted against increasing standard deviation as shown in Figure A4.1. The restriction placed on the inputs were that the values were bounded between the values 0 and 1 (Unclipped) or rounded, as before, to the nearest value, 0 or 1.

4. Selective feedback: This function places the corrupted image vector (from the input data file) into the array and feeds back the output of the array, overwriting the initial image. This was to evaluate the ability of the array to reconstruct the corrupted image to the nearest trained image. This mode does not allow partial feedback of the array output but does allow for a larger array input than input image, the excess being initially zeroed.

5. Hopfield Style test: This is the main feedback mechanism used when exercising a trained array. On each exercising trial, a selectable

proportion of the array output is fed back to the input n times, overwriting that section of the initial input. This is shown in Figure A4.2.

A4.6 Graphical Facilities

The program offers the following graphical facilities. It can produce a block plot of the element weight connections. This is shown in Figure A4.3. and shows the connection weights for each element. Each block represents an element. Each square in the block represents a weight, red for excitatory and black for inhibitory. The magnitude of the weight is indicated by the area of the square where the maximum area is scaled to the maximum weight to be plotted in the diagram and is listed at the base of the figure. The position of the square in the block indicates to which element that weight connects.

Since this particular example is for the Hopfield Network, as explained in Chapter Five, there will be one blank square representing null connection from the element to itself. In this case, also, weights are symmetric, eg. the connection of the top left element to the bottom right element, represented by the bottom right square in the top left block, is the same value as that shown by the top left square in the bottom right block. The small square at the top left of each block represents the connection to a virtual element which is always on, ie. the stimulus. A line plot is also available, shown in Figure A4.4 which instead uses red and black arcs to connect elements together. At each element, there are two local arcs, the upper one represents the connection of the element to the coordinate immediately above it in the layer.

For the Hopfield Network, this corresponds to the elements connection to itself and therefore is not shown in this plot. The lower arc corresponds to the elements connection to its stimulus. The relative thicknesses of the lines are intended to convey the respective magnitudes of the connections. The program can also plot the weights on a graph for selected elements. Figure A4.5 shows the weights for one of the second level elements in a 5 x 5 x 2 array using the gradient following A_{R-P} method, learning to recognise and reproduce the figure zero. The program can also easily output the input–output vector for this case, as shown for the previous example, in Figure A4.6. The global reinforcement may also be plotted, as shown for the A_{R-P} Exclusive Or problem in Figure A4.7.

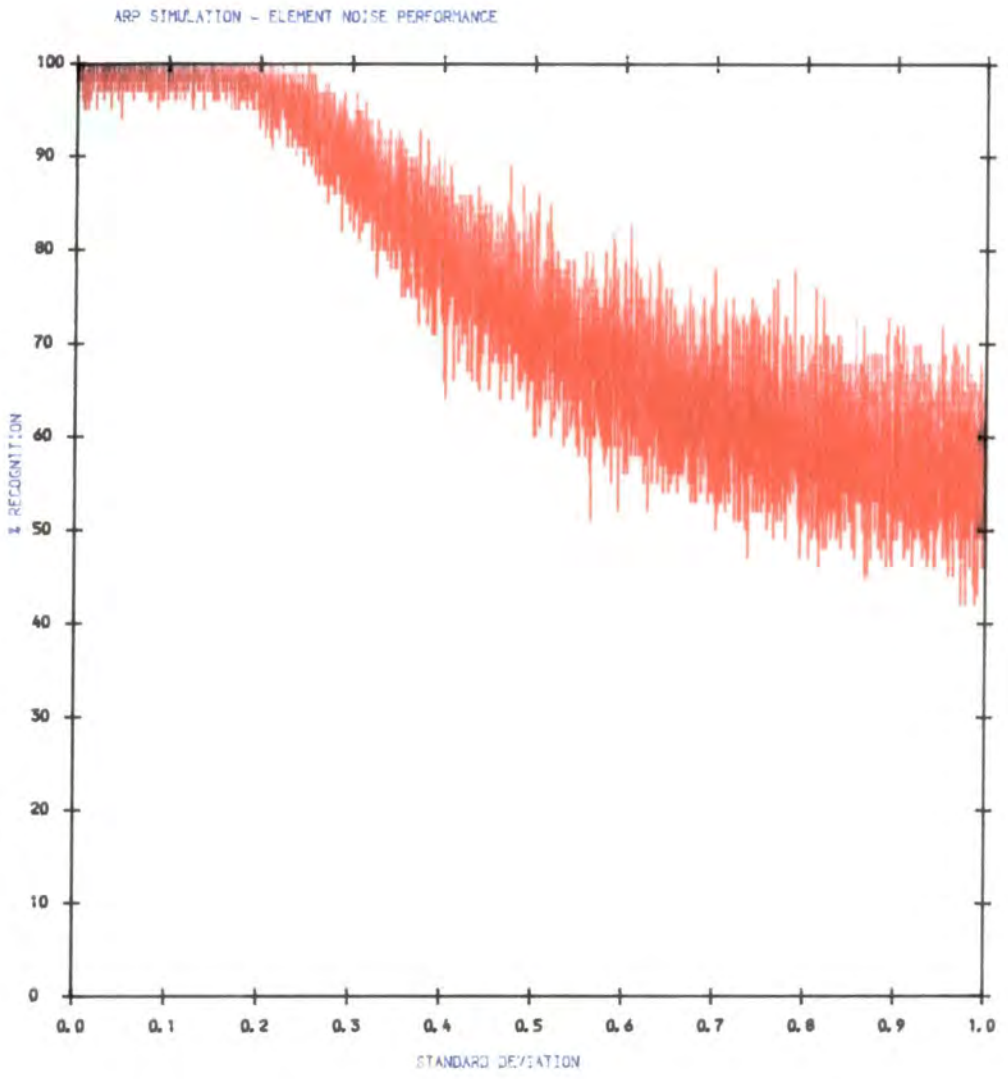


Figure A4.1 – Noise response example for EXOR problem – A_{R-P} network

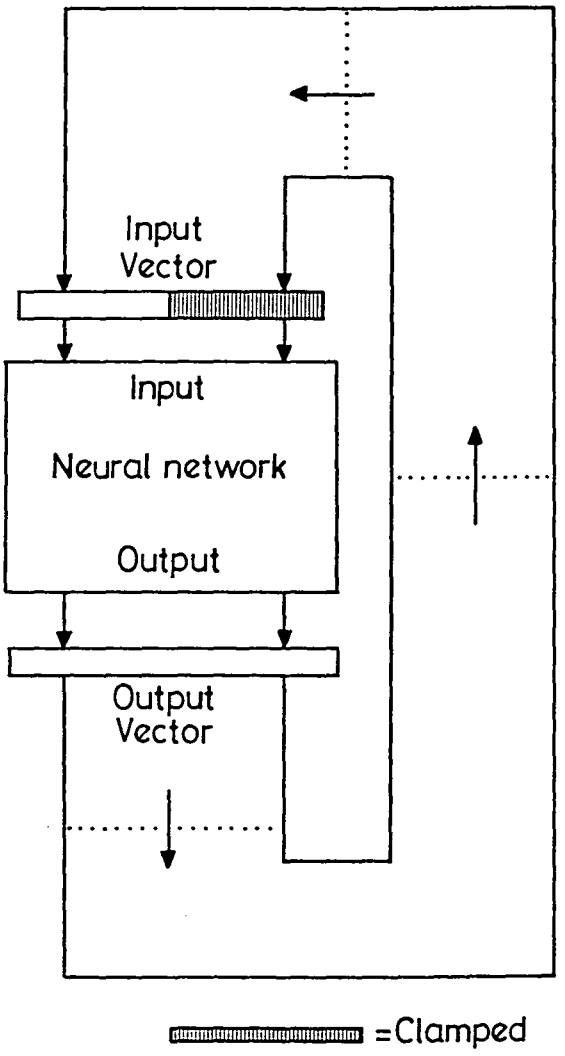


Figure A4.2 – Hopfield Style feedback arrangement

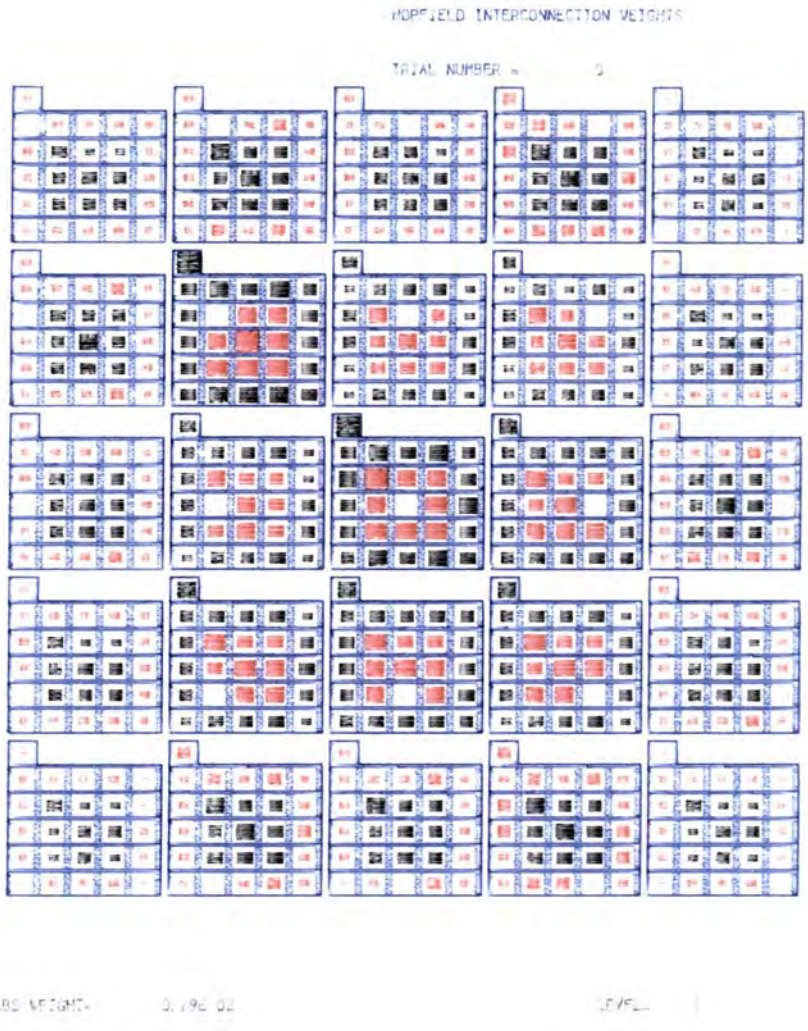


Figure A4.3 – Block plot example using Hopfield 5 x 5 network

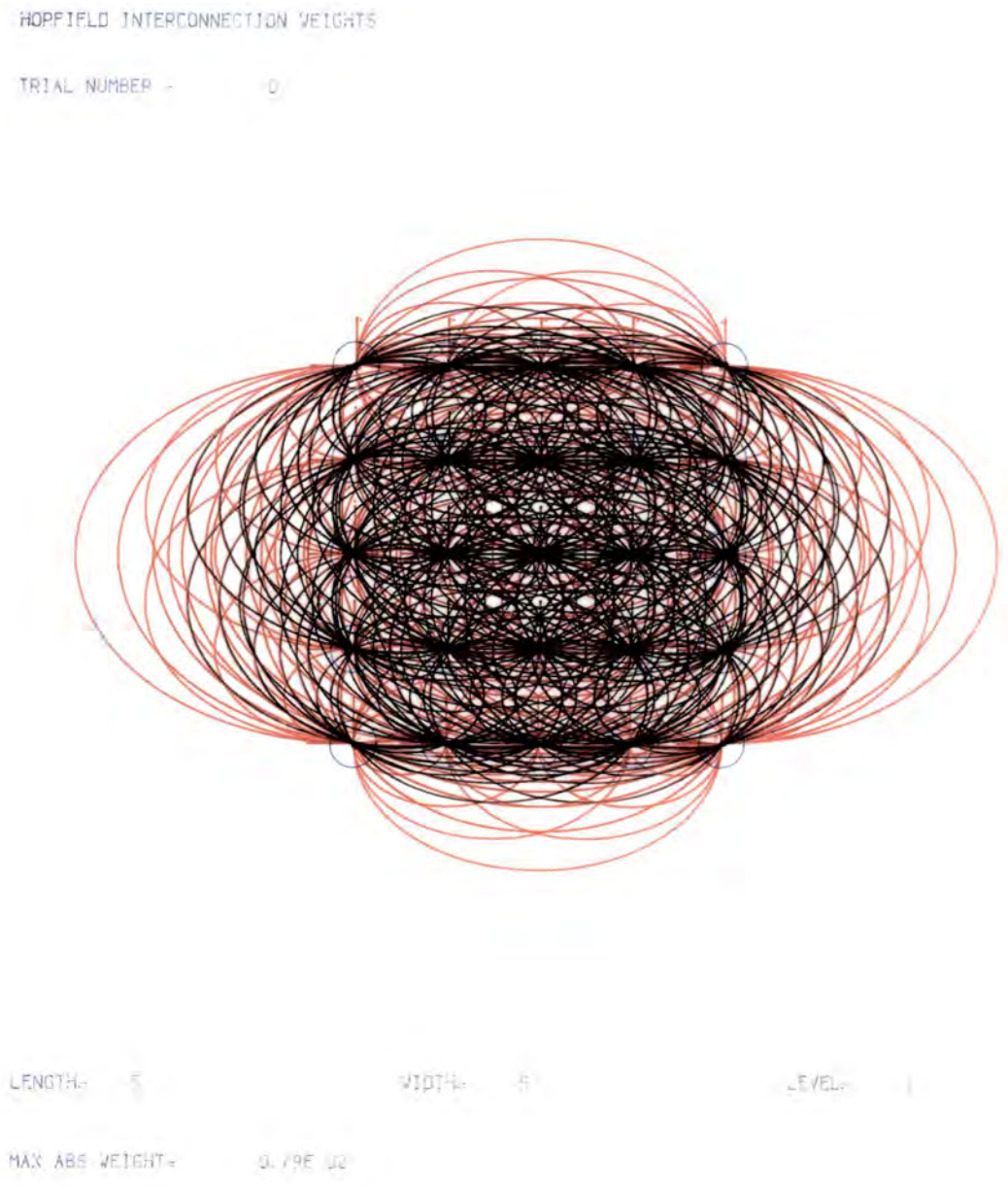


Figure A4.4 – Line plot example using Hopfield 5 x 5 network

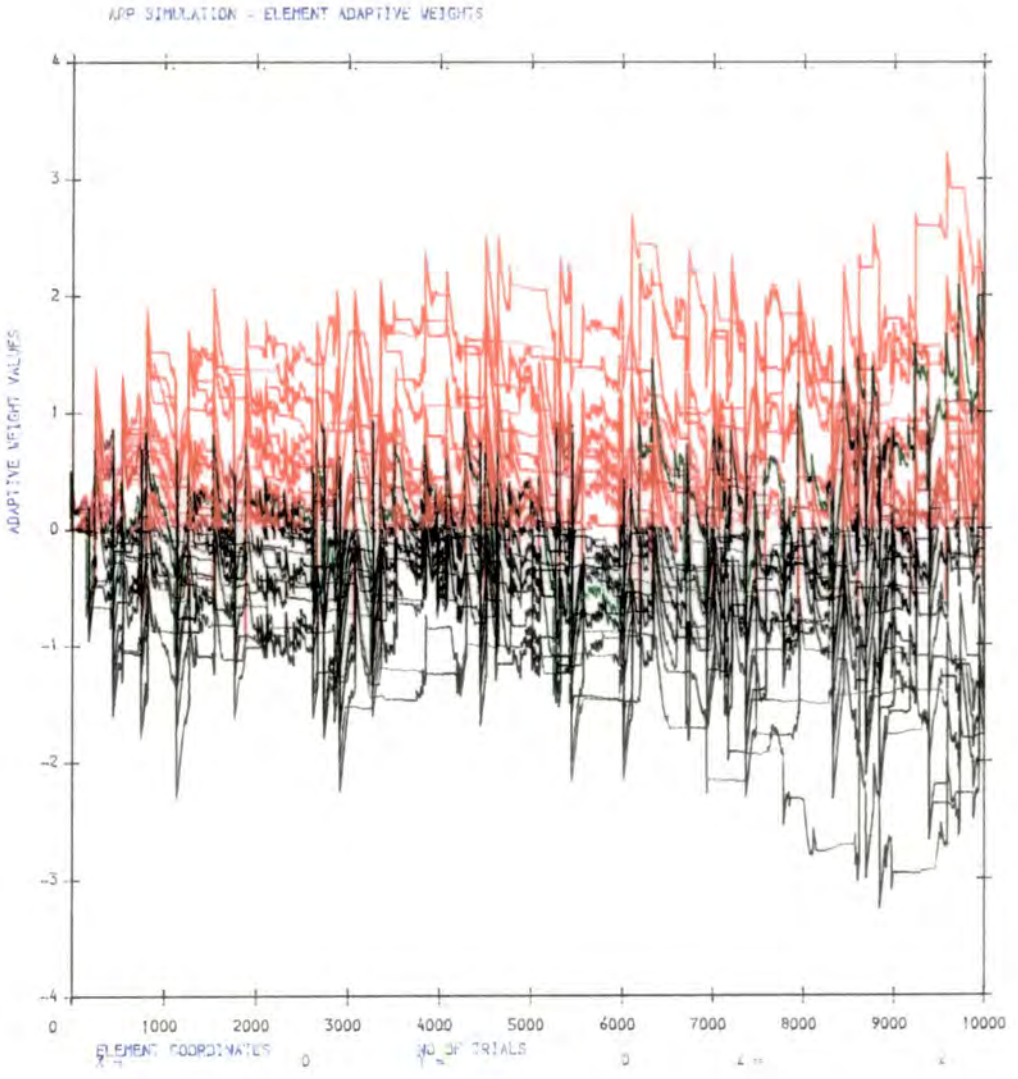


Figure A4.5 – Weight plot example using gradient A_{R-P} network

Figure A4.6 – Input–Output vector depiction for figure zero recognition

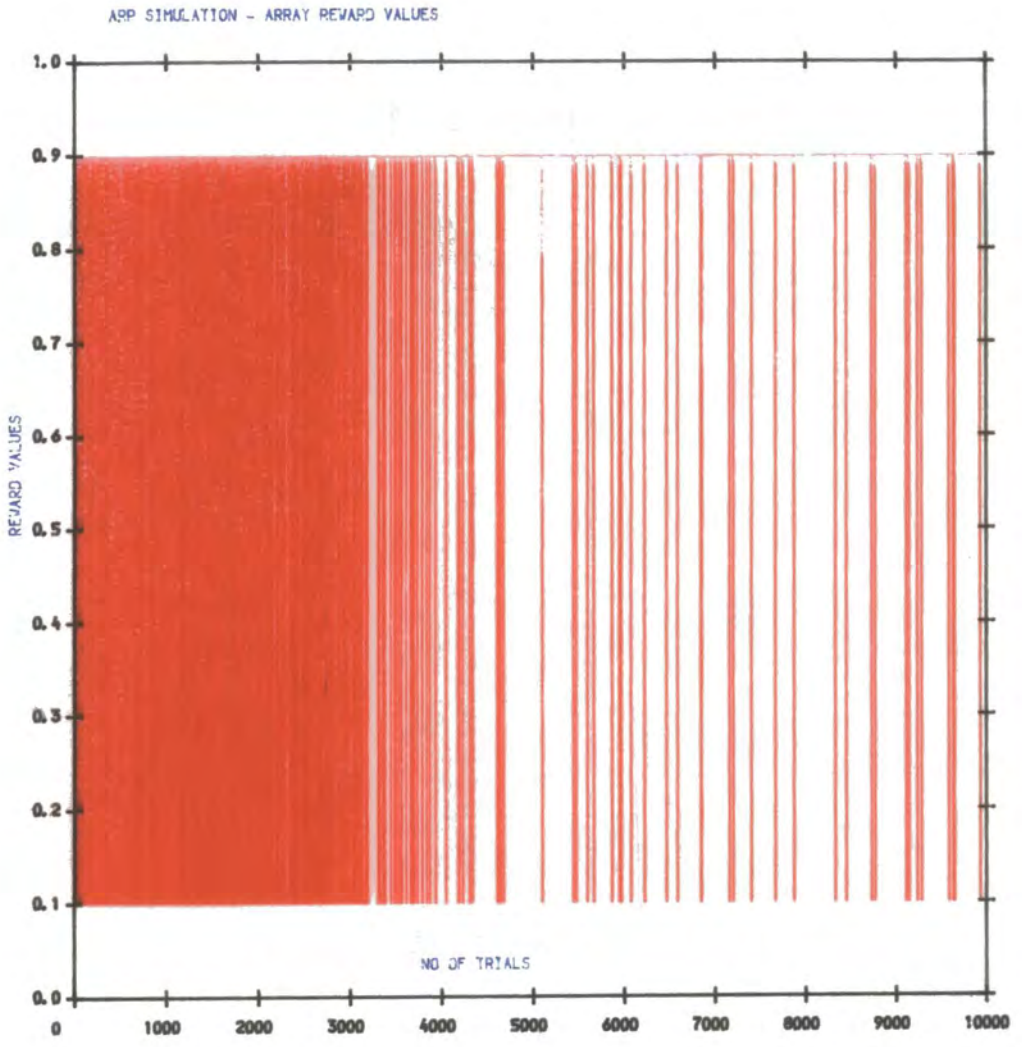


Figure A4.7 – Global reinforcement example **EXOR** problem – A_{R-P} network

Appendix Five

Simulation programs – simple A_{R-P} networks

A5.1 Introduction

This appendix briefly documents the programs (included in Volume 2), necessary to perform the simple one and two element A_{R-P} simulations.

A5.2 One element A_{R-P} simulation program

Program ARP.1EL simulates one A_{R-P} element for three values of the learning parameter λ and uses a NAG library random number generator. The program calls a subroutine ARP.SUB which describes the operation of the actual A_{R-P} element. A function PSI is defined in ARP.FUNC which is simply the saturating exponential output function of the element and is used to compute M_t. The topology of the simulation is shown in Figure A5.1. and the environmental reward probabilities in Table A5.1.

A5.3 Two element A_{R-P} simulation program

Program ARP.2EL simulates two linearly connected A_{R-P} elements for three values of the learning parameter λ using ARP.SUB and ARP.FUNC as previously described. The topology of the simulation is shown in Figure A5.2. and the environmental reward probabilities in Table A5.1.

A5.3 Two element Exclusive OR A_{R-P} simulation program

Program ARP.EXOR simulates two semi-parallel A_{R-P} elements for one value of the learning parameter λ using ARP.SUB and ARP.FUNC as previously described. The topology of the simulation is shown in Figure A5.3. and the environmental reward probabilities in Table A5.2.

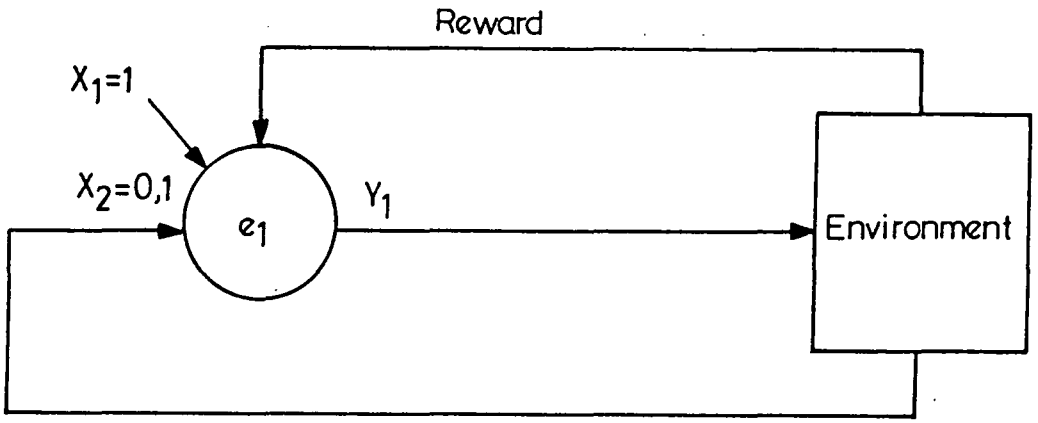


Figure A5.1 One A_{R-P} element topology

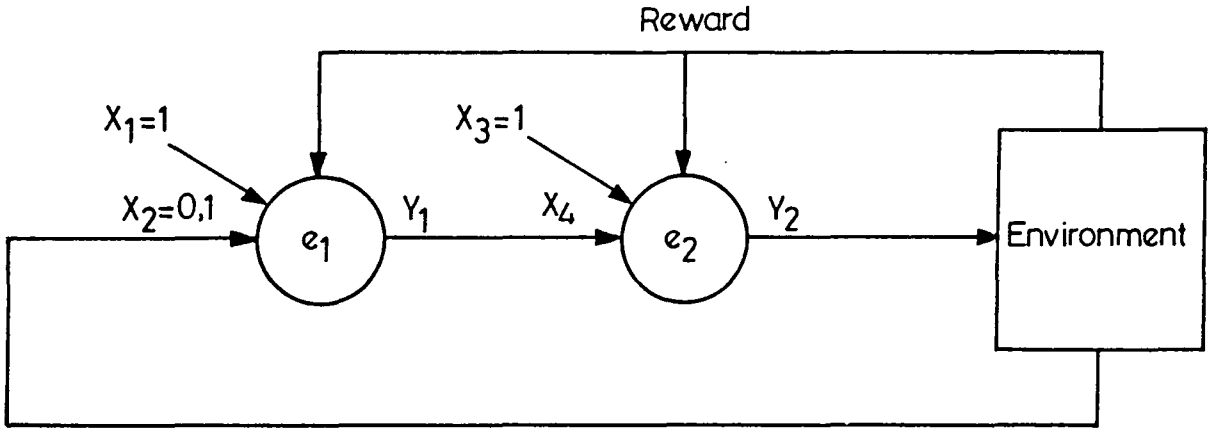


Figure A5.2 Two A_{R-P} element (linear) topology

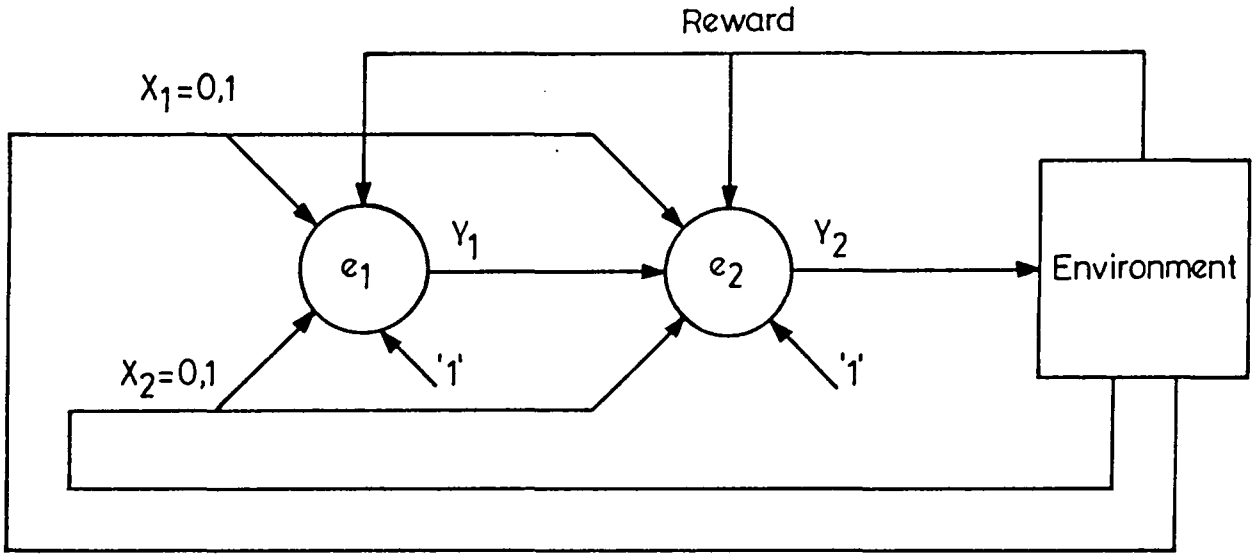


Figure A5.3 Two A_{R-P} element (Exclusive OR) topology

Table A5.1 - Reward Probabilities - Example 1

x	d(x, +1)	d(x, 0)
(1,0)	0.1	0.9
(1,1)	0.9	0.1

Table A5.2 - Reward Probabilities - Example 2

x	d(x, +1)	d(x, 0)
(0,0)	0.1	0.9
(0,1)	0.9	0.1
(1,0)	0.9	0.1
(1,1)	0.1	0.9

Appendix Six

Derivation of the asymptotic values for the A_{R-P} element simulations

A6.1 Introduction

This appendix presents a mathematical method for predicting the A_{R-P} convergence values for given Reward Probabilities and learning rate λ for one element. A computer program is included in Volume 2.

A6.2 Analysis

From Barto et al, [107],

Let the probability of presentation of a particular x vector be

$$P(x_t = x) = \xi_x \qquad (A6.1)$$

where x_t denotes a particular x vector at time t.

Let the probability of the element having action a_t at time t, given x_t be;

$$P\{a_t = a \mid x_t = x\} = P_t^{ax}(s_t) \qquad (A6.2)$$

where

$$s_t = \sum_{n=1}^{N} x_n(t).w_n(t) \qquad (A6.3)$$

Let the probability of the environment having response b_t at time t, given a_t and x_t be;

$$P\{b_t = b \mid x_t = x \ ; \ a_t = a\} = d_{xa} \qquad (A6.4)$$

The weight update is given by the A_{R-P} algorithm as;

$$\Delta_w = -\lambda_{b_t}\ \rho_t\ x_t\ [E(a_t \mid x_t) - a_t b_t] \tag{A6.5}$$

The performance at time t, M_t is given by;

$$\begin{aligned} M_t &= \sum_x \xi_x P\{b_t = 1 \mid x_t = x\} \\ &= \sum_{x,a} \xi_x P_t^{ax}(s_t) d_{ax} \end{aligned} \tag{A6.6}$$

At convergence, the expectation of an update value for all x is zero, ie;

$$E\{\Delta_w \mid x_t = x\} = 0 \tag{A6.7}$$

and so, given that the expectation operator is defined as;

$$E(X) = \sum_x x.P(X = x) \tag{A6.8}$$

then equation A6.7 becomes;

$$E(-\lambda_{b_t} \rho_t x_t [E(a_t \mid x_t) - a_t b_t]) = 0 \tag{A6.9}$$

for all x_t. In the simulations performed, ρ_t is taken as constant. Therefore, dividing both sides by $-\rho_t$;

$$\sum_{a,b} \{\ \lambda_{b_t}\ [E(a_t \mid x_t) - a_t b_t]\}\ \ P_t^{ax}(s_t) d_{ax} = 0 \tag{A6.10}$$

Note that the probability of the element having action $a_t = +1$ is given by;

$$P_t^{+1x} = \frac{1}{1 + e^{-s_t/T}} \tag{A6.11}$$

and that the expectation of an action a_t, $E(a_t \mid x_t)$ is given by;

$$\begin{aligned} E(a_t \mid x_t) &= 1.P_t^{+1x} - 1.P_t^{-1x} \\ &= 1.P_t^{+1x} - 1.(1 - P_t^{+1x}) \\ &= 2.P_t^{+1x} - 1 \end{aligned} \tag{A6.12}$$

Therefore, the expectation of an update, given by equation (A6.10) becomes;

$$\sum_{a,b} \left\{ \lambda_{b_t} \left[\sum_a a_t P_t^{ax} - a_t b_t \right] \right\} P_t^{ax}(s_t) d_{ax} = 0 \qquad (A6.13)$$

Since the element has only two possible output values for a_t, this equation is quadratic in P_t^{+1x} for each x and may be solved for s_t. By then substituting into equation (A6.6), it is possible to determine the asymptotic value of M_t. Since for the two input single element case, there are two input x vectors, there will be two simultaneous equations to solve in order to determine the asymptotic element weights.

Expanding equation (A6.13) for both values of element output yields;

$$\sum_b \left\{ \begin{array}{l} \lambda_{b_t} \left[2.P_t^{+1x} - 1 - b_t \right] P_t^{+1x}(s_t) d_{+1x} \\ \\ \quad + \left[2.P_t^{+1x} - 1 + b_t \right] (1 - P_t^{+1x}(s_t)) d_{-1x} \end{array} \right\} = 0 \qquad (A6.14)$$

At this stage, it becomes convenient to abbreviate some terms; Let $P_t^{+1x} = p$, $d_{+1x} = d$ and $d_{-1x} = e$. Then equation (A6.14) becomes;

$$\sum_b \{ \lambda_{b_t} \left[2.p - 1 - b_t \right] p.d + \left[2.p - 1 + b_t \right] (1 - p).e \} = 0 \qquad (A6.15)$$

Furthermore, note the values which b_t may take; (Using a rough and ready representation which refers to the probability range in which a trial of the the environment action probability falls as p_{env} and the corresponding range for the element as p_{elem}).

$$b_t = \begin{cases} +1 & \text{if } p_{elem} = p \text{ and } p_{env} = d \\ & \text{or } p_{elem} = (1-p) \text{ and } p_{env} = e \\ -1 & \text{if } p_{elem} = p \text{ and } p_{env} = (1-d) \\ & \text{or } p_{elem} = (1-p) \text{ and } p_{env} = (1-e) \end{cases} \qquad (A6.16)$$

Note also that;

$$\lambda_{b_t} = \begin{cases} +1 & \text{if } b_t = +1 \\ \lambda & \text{otherwise} \end{cases} \qquad (A6.17)$$

Then equation (A6.15) becomes;

$$[2.p-1-1]\ p.d+\lambda[2.p-1+1]\ p.(1-d)$$
$$+[2.p-1+1]\ (1-p).e+\lambda[2.p-1-1]\ (1-p).(1-e)=0 \qquad (A6.18)$$

Expanding this and dividing throughout by 2 gives;

$$[p-1]\ p.d+\lambda[p^2]\ (1-d)+$$
$$[1-p]\ p.e+\lambda[p-1][1-p]\ (1-e)=0 \qquad (A6.19)$$

Multiplying out gives;

$$p^2.d-p.d+\lambda.p^2-d.\lambda.p^2+p.e-p^2.e+\lambda[2p-p^2-1)][1-\epsilon]=0 \qquad (A6.20)$$

Therefore collecting terms in p gives;

$$p^2[(d-e)(1-\lambda)]+p[2\lambda(1-e)+\epsilon-d]+\lambda(e-1)=0 \qquad (A6.21)$$

The roots of this equation give values of p for which equation A6.5 is zero. Asymptotes calculated for Example 1 as shown in Table A6.1, where $x^{(1)}=(1,0)$ and $x^{(2)}=(1,1)$. In this case,

$$i=1$$
$$d=0.1$$
$$e=0.9 \qquad (A6.22)$$
$$\lambda=0.25$$

Then substituting into equation (A6.21) we obtain the quadratic:

$$-0.6p^2+0.85p-0.025=0 \qquad (A6.23)$$

Using the standard method for solving a quadratic equation of the form;

$$ax^2+bx+c=0 \qquad (A6.24)$$

whose roots are given by;

$$x = \frac{-b \pm \sqrt{b^2 - 4(ac)}}{2a} \tag{A6.25}$$

Therefore the solutions of equation (A6.23) are;

$$\begin{aligned} P &= 0.03 \\ P &= 1.3866 \end{aligned} \tag{A6.26}$$

Clearly P is constrained as $0 \leq P \leq 1$ and so only the solution P=0.03 is valid. Similarly, for i=2;

$$\begin{aligned} i &= 2 \\ d &= 0.9 \\ e &= 0.1 \\ \lambda &= 0.25 \end{aligned} \tag{A6.27}$$

Then substituting into equation (A6.21) we obtain the quadratic:

$$0.6p^2 - 0.35p - 0.225 = 0 \tag{A6.28}$$

The solutions of equation (A6.28) are;

$$\begin{aligned} P &= 0.9699 \\ P &= -0.3866 \end{aligned} \tag{A6.29}$$

Clearly P = 0.9699. We therefore have the following result;

$$\begin{aligned} P_t^{+1x^{(1)}}(s_t) &= 0.03 \\ P_t^{+1x^{(2)}}(s_t) &= 0.9699 \end{aligned} \tag{A6.30}$$

From equations (A6.2) and (A6.3);

$$P_t^{+1x^{(1)}}(s_t) = \frac{1}{1 + e^{-(w_1 + w_2)/T}} \tag{A6.31}$$

$$P_t^{+1x^{(2)}}(s_t) = \frac{1}{1 + e^{-(w_1)/T}} \tag{A6.32}$$

Solving for w_1 and w_2 gives;

$$\begin{aligned} w_1 + w_2 &= 1.737 \\ w_1 &= -1.737 \end{aligned} \tag{A6.33}$$

Therefore giving the weights as;

$$\begin{aligned} w_1 &= -1.737 \\ w_2 &= 3.474 \end{aligned} \tag{A6.34}$$

From equation A6.6, M_t is given by:

$$M_t = \sum_{x,a} \xi_x P_t^{ax}(s_t) d_{ax} \tag{A6.35}$$

and so;

$$\begin{aligned} M_t = \quad & 0.5 * 0.03 * 0.1 \\ & + 0.5 * (1 - 0.03) * 0.9 \\ & + 0.5 * 0.9699 * 0.9 \\ & + 0.5 * (1 - 0.9699) * 0.1 \\ = \quad & 0.876 \end{aligned} \tag{A6.36}$$

Table A6.1 - Reward Probabilities - Example 1

x	d(x, +1)	d(x, 0)
(1,0)	0.1	0.9
(1,1)	0.9	0.1

SECOND INTERNATIONAL CONFERENCE on

Paper 29

LASER ANEMOMETRY -
ADVANCES AND APPLICATIONS
21st to 23rd September 1987, Strathclyde, UK.

APPLICATION OF A STOCHASTIC SYSTOLIC ARRAY TO PHOTON CORRELATION AND LDA TECHNIQUES

R.A.Leaver, P.Mars, M.Sarhadi

School of Engineering and Applied Science,
Durham University,
South Road,
Durham DHI 3LE,
UK.

Abstract

This paper suggests the potential application of a Stochastic Systolic array to Laser Doppler Anemometry (LDA) Photon Correlation techniques. The array, currently being designed in VLSI at Durham performs the computation of the Complex Correlation function and Discrete Fourier Transform.

This method exhibits low latency, each cell in the array is of simple, regular and potentially fault tolerant design enabling high speed operation. As is characteristic of stochastic methods, the accuracy is time rather than device dependent thereby offering a fast estimation facility together with useful feedback and control properties for adaptive processing. Some example results revealing the system's resilience to noise are included.

Organised by LDA USERS GROUP.
Sponsored by University of Strathclyde.
Co-sponsored by the Royal Aeronautical Society
and the Institution of Mechanical Engineers

Introduction

Stochastic processing utilises probability as the medium for performing the conventional computation operations of multiplication and addition. We have previously described the the application of stochastic techniques for performing Cyclic Correlation and the Discrete Fourier Transform, for use in Image Processing and Spectral Analysis[1].

In this paper, we examine the potential application of the Stochastic Systolic array to Photon Correlation in applications such as Laser Doppler Anemometry. A description of the array is given, together with a brief introduction to the LDA signal processing requirement. Results are presented which demonstrate the stochastic array's inherent resilience to noise with a Gaussian profile and zero mean. The array has the dual advantage that it can take the direct output bit stream from the stochastic encoding process such as a photomultiplier tube and either perform computation of the Cyclic Complex Correlation function or the Discrete Fourier Transform, providing direct Spectral Analysis of the results. The paper concludes with a brief description of the expandable array VLSI chip being designed at Durham.

Laser Doppler Anemometry

The fundamental process involved here is one of Photon Correlation[2]. The output pulses from a Photomultiplier tube are autocorrelated against variable time delayed sequences in order to evaluate the periodicity inherent in the incident light. Alternative methods include using a sweeping or tracking frequency filter. The number of pulses in any given time span is related to the amplitude of the incident light via a Poisson Distribution. The greater the amplitude, the more pulses are generated.

Stochastic Processing

Stochastic techniques involve the representation of values by the probability of occurrence of a pulse in a bit serial pulse train[3]. Stochastic encoding is accomplished by thresholding the analogue deterministic value using a one bit comparator with a random threshold where each random value is equiprobable, ie. generating a Bernoulli sequence. Care must be taken that the random level generation processes used for encoding have a zero cross correlation component. In this way, information may be transmitted using independent single bits. This introduces the disadvantage that we can no longer use a logarithmic representation, which reduces the effective dynamic range of the system. However, the principal advantage of the system is that because each bit in the pulse stream is independent, accuracy is determined primarily by the time taken to form the average. As in standard probability, as this time interval tends to infinity, the result tends to the exact deterministic value. We may therefore choose to take results at any time from the array although the greater the averaging time interval, the more accurate the result. This precision acquisition with time, rather than as a feature of the architecture is the predominant characteristic of the stochastic approach. It should be mentioned here that we define the array performance for input values encoded using the Bernoulli (ie. equiprobable) Distribution, and the output of the array should be interpreted accordingly for inputs encoded using any other distribution as in the LDA application.

Stochastic Computation Elements

Stochastic computation elements are almost trivially simple to implement. They vary according to the mapping used to convert the deterministic value to a stochastic representation. In the mapping adopted for the correlator, we choose to use a dual line bit serial representation where positive values are expressed on one line, negative on the other. Multiplication is performed using four AND gates, Addition requires 2 OR gates. For addition, a simple corrective circuit is required to register the coincident pulse case and then compensate the bit stream by reinserting a pulse. Preliminary simulations indicate that with slight overscaling of the input, a simple 4 state up/down counter arrangement is sufficient for accurate addition. This is the stochastic equivalent of the CARRY operation in deterministic addition.

Output Interface

In the simplest case, the output interface accumulates the number of pulses present and then averages by dividing by the total number of slots available. This is adequate for stationary inputs, but much past research has involved study of tracking non stationary ie. dynamic inputs[4]. The resulting output interfaces, named 'ADDIEs' utilise various alternative forms of averaging in order to provide fast convergence to an accurate answer in the stationary case and also to provide a fast response to a change in input for non-stationary inputs. The array described feeds the output bit stream into a shift register whose length is that corresponding to the required time interval. The number of pulses within the register is counted and the average formed by dividing by the length of the register. Since the register length is a power of two, the division is a simple binary shift operation.

Systolic Arrays

Systolic arrays were first described by Kung and Leiserson[5] and enable very efficient parallel computation of many common processing problems. The systolic array consists of a series of identical cells, each performing the same operation, communicating only with nearest neighbours, on each clock cycle. Since the array is regular in design with identical elements, the design and test process is simplified. Also because of the local, synchronous connections, the array may be operated at high speeds. Deterministic systolic array correlators have been designed for use in various signal processing applications[6].

Stochastic Systolic Correlator

The computation of any N point transform such as Cyclic Correlation or the Discrete Fourier Transform requires at least N and up to N^2 operations, depending on the algorithm used. Defining Cyclic Correlation as:

$$y_n = \sum_{k=0}^{N-1} h_k . x_{<n+k>} \tag{1.0}$$

where

$$n = 0,1..N\text{-}1$$

and

$$< > = \text{reduction MOD N}$$

The convolution equation is the same except that <n-k> instead of <n+k>.

Expanding this out for N = 3 gives:

$$y_0 = h_0.x_0 + h_1.x_1 + h_2.x_2$$

$$y_1 = h_0.x_1 + h_1.x_2 + h_2.x_0 \qquad (1.1)$$

$$y_2 = h_0.x_2 + h_1.x_0 + h_2.x_1$$

Applying as horizontal row inputs, the coefficients h_0, h_1, h_2 applied to nodes 9,5 and 1 respectively of the array in Figure 1 and as vertical column inputs, the coefficients x_0, x_1, x_2, x_0, x_1 applied to nodes 1,3,6,10 and 13 respectively (where x_0 and x_1 are repeated to generate the correct expressions along the diagonals. Then it may be seen that with zero input to the diagonal nodes 10,11 and 12, and the action of each cell as shown in Figure 1, the product/accumulate along the diagonals will produce the values y_0, y_1, y_2 at the diagonal nodes 1,2 and 3. In our stochastic approach, we have deliberately used an N x N array performing N^2 operations. The advantage gained is that each stochastic cell is of simple compact design performing a one bit operation every clock cycle. In this arrangement, each cell is fully utilised at all times. For the initial data vector, at least N cycles are required to pass the stochastic values through the array before averaging of the first output vector takes place. Results may then be taken at any time depending on the required accuracy. By using four identical arrays, complex values may be accomodated, enabling computation of the Discrete Fourier Transform using the well known approach of Rader[7] thus enabling fast spectral analysis.

Stochastic Simulation Results

The operation of the array has been simulated for both Complex Correlation and the Discrete Fourier Transform using a FORTRAN program. Computational overheads have restricted the simulation to values of N less than 50 points. This has also precluded study of multidimensional mappings of small N stochastic blocks to perform large N transforms. The output interface is the simple averaging procedure for stationary inputs. Figure 2a shows the real and imaginary outputs from the array for a 6 point Complex Cyclic Correlation with zero noise. This shows the value of each output point in the correlation against the number of clock cycles. (after N initial cycles). The averaging process is being performed in a shift register of capacity 2048 and so the averaging time interval is 2048 slots. The dotted lines indicate the theoretical answers to which the array outputs should asymptote. Figure 2b shows an expanded region of the same plot. Figure 2c shows an identical simulation run (using the same random number sequence) but this time with a significant amount of noise on the inputs. We selected a noise level of 3 dB where this is defined as the dB ratio of the Input Dynamic Range of the system (in this case 5.0) to the Standard Deviation of the injected noise. The noise has a Gaussian Distribution and zero mean. The averaging process inherent in stochastic computation as expected, effectively removes this. Figure 2d shows an expanded region of the same plot. The reference values have been chosen such that the correlation is performing the first stage in computing a 7 point Discrete Fourier Transform of the input data sequence via Rader's Theorem. Any M point transform may be accomodated via an $N = M - 1$ point Cyclic Complex Correlation as long as M is an odd prime.

Consequently the reference sequence consists of reordered sine and cosine ('twiddle') terms. The system scale has been chosen to be 5.0 in order to encompass the maximum output value of the correlation and interim noise.

Implementation

The systolic array as described is being designed at Durham by two of the author's colleagues at Durham[8]. The chip contains an extremely compact 6x6 cell array plus interface circuitry. The number of cells has been chosen to be N = 6 in order to fit in with the majority of the computer simulations. The chip is designed to be expandable, ie. easily connectable to identical chips to make up a larger array, with no additional logic. The up/down counter which is part of the stochastic addition process is being efficiently implemented as a Finite State Machine. The number of pins required grows rapidly for larger N and it is expected that a practical large array will be made from small sub-blocks using surface mount or similar technology. The inherent simplicity of the array is such that fault tolerance and redundancy is easily built in, which may imply potential Wafer Scale implementation. In an array made up from such small sub-blocks each of size N, we hope to be able to investigate the advantages of global broadcasting of the data and reference sequences. This should enable a change in input to be entered at each sub-block, instead of 'rippling' through the macro-array. Care will be needed to ensure the independence of each sequence by use of suitable one-bit delays where necessary.

Conclusions

Simulation results indicate that conservative correlation accuracies of 6 - 8 bits relative to the system dynamic range, can be achieved within 500 clock cycles with the array, as measured against a stationary input vector. The accuracy of the system when used to perform the DFT may also be substantially increased, depending on the data values due to the mixed deterministic/stochastic nature of the computation. The system can therefore provide fast Spectral Analysis of dynamically changing inputs with accuracy increasing over larger averaging intervals. The tradeoff is always one of Accuracy vs Bandwidth. The system is extremely noise tolerant and should be able to accept direct one bit inputs from devices such as photomultipliers, to perform efficient LDA analysis.

Summary

We have suggested a novel application of a stochastic systolic array for use in Photon Correlation and Laser Doppler Anemometry. Results may be taken at any time using an output averaging interface with selectable characteristics.

Acknowledgements

One of us (R.A.L) would like to acknowledge the support of British Aerospace PLC and an SERC Industrial Studentship.

References

1. Leaver, R.A.: 'Application of Stochastic techniques to Systolic Array Computation'. Durham University, School of Engineering and Applied Science (Internal Document), 1986, May.

2. Abbiss, J.B., Chubb, T.W., and Pike, E.R.: 'Laser Doppler Anemometry'. Optics and Laser Tech.,1974, December.pp.249-261

3. Mars, P., and Poppelbaum, W.J.: 'Stochastic and Deterministic Averaging Processors'. IEE Digital Electronics and Computing Texts., (Peter Peregrinus 1981)

4. Miller, A.J., Brown, A.W., and Mars, P.: 'Optimum Criteria for output interfaces in digital stochastic computers'. Electron. Letters., 1975, 11, pp.326-327

5. Kung, H.T., and Leiserson, C.: 'Systolic Arrays for VLSI' in 'Introduction to VLSI Systems'.Section 8.3., (Addison Wesley 1980)

6. McWhirter, J.G., and McCanny, J.V.: 'A Novel Multi-bit Convolver/Correlator Chip Design based on Systolic Array Principles'. Proc SPIE., 1982, Vol 341, Paper 08.

7. Rader, C.M.: 'Discrete Fourier Transforms when the number of samples is prime'. Proc IEEE, 1968, 56, 6, pp 1107-1108

8. Taylor, S., and Hartmann, J.: 'A Chip Design to Perform Systolic Array Computations'. Durham University, School of Engineering and Applied Science (Internal Design Document), 1987, March.

9. Davenport, W.B.: 'Probability and Random Processes'., (McGraw-Hill 1970)

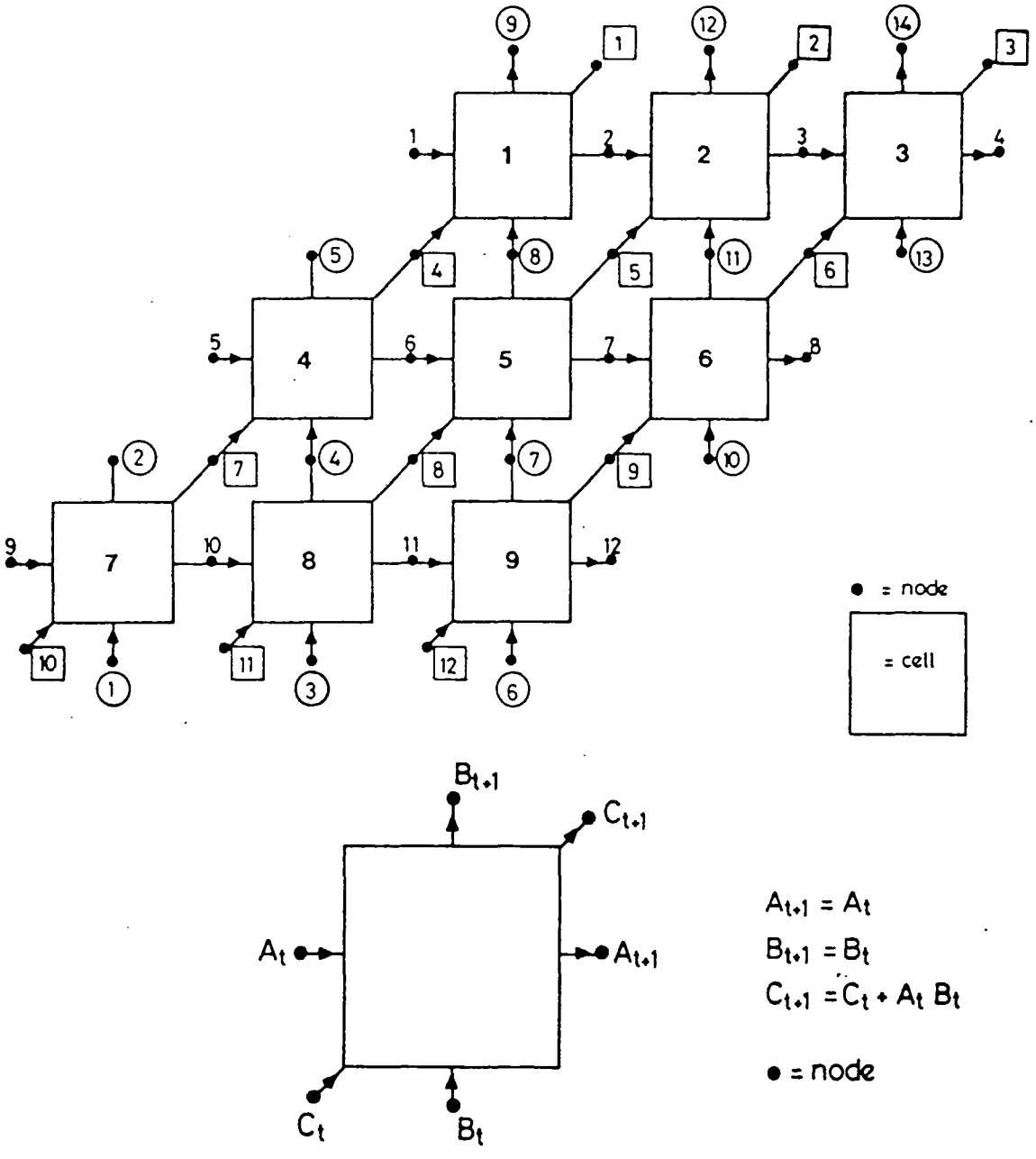
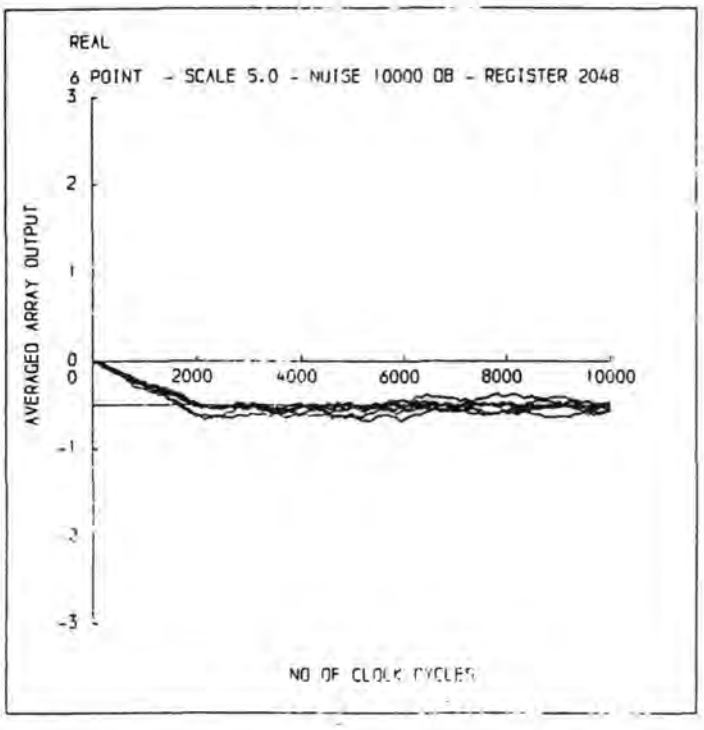


Figure One : 3 x 3 Stochastic Systolic Array



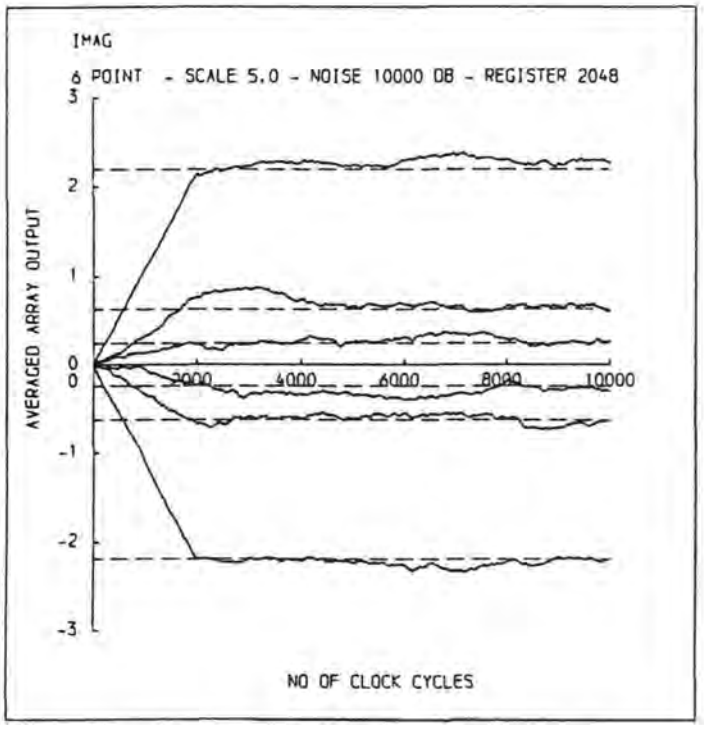


Figure Two (a) : Example array output with zero noise input.

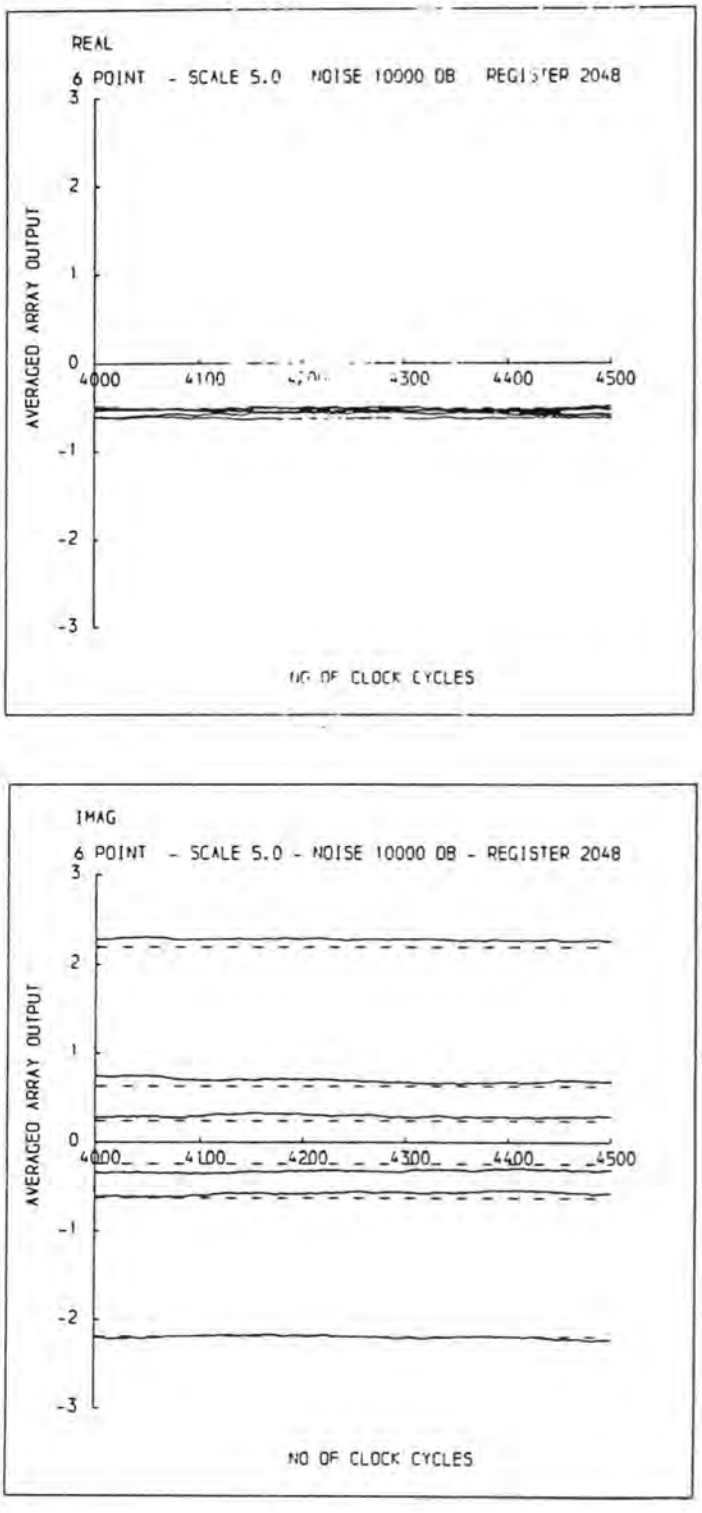
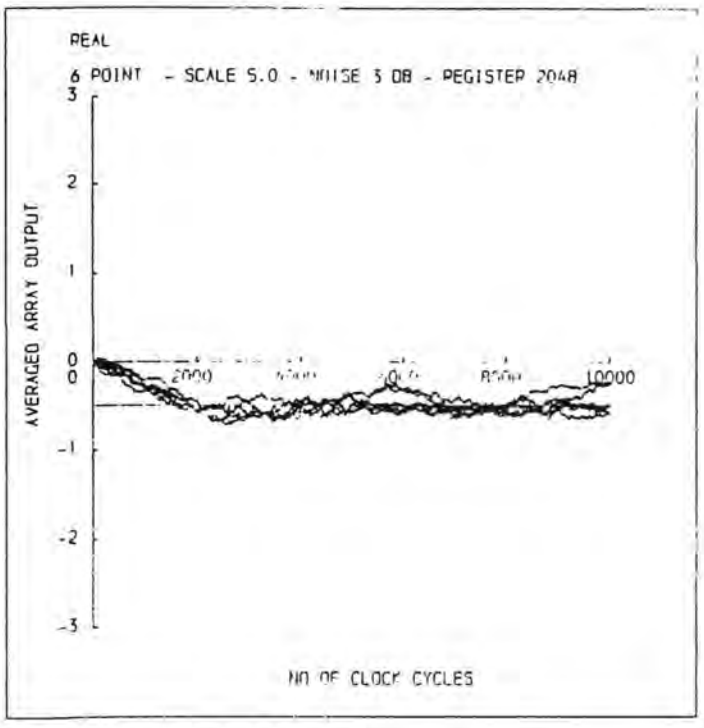


Figure Two (b) : Example array output with zero noise input.



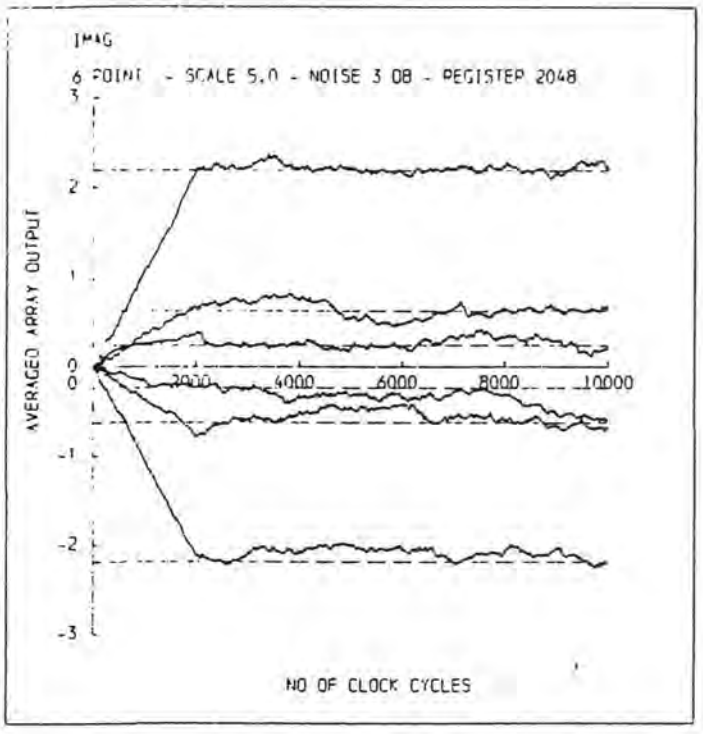
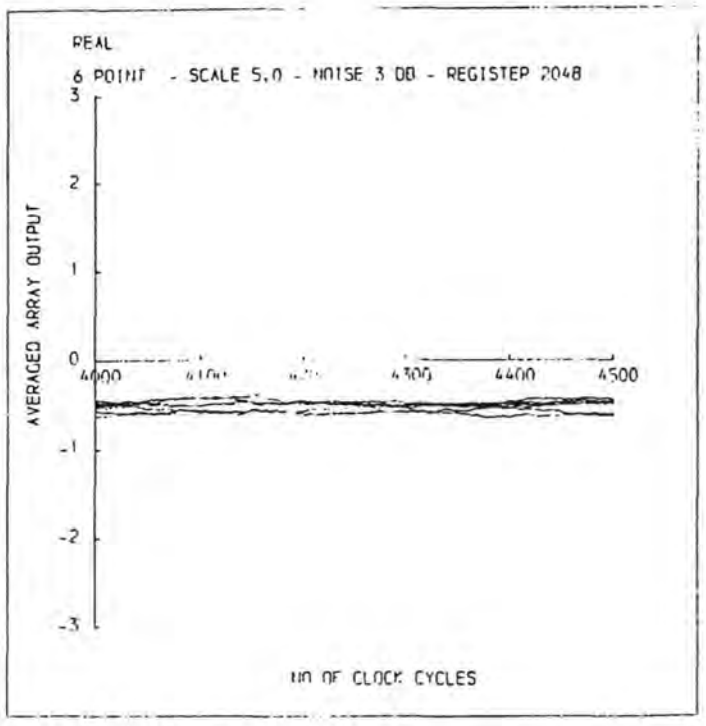


Figure Two (c) : Example array output with 3dB noise input.



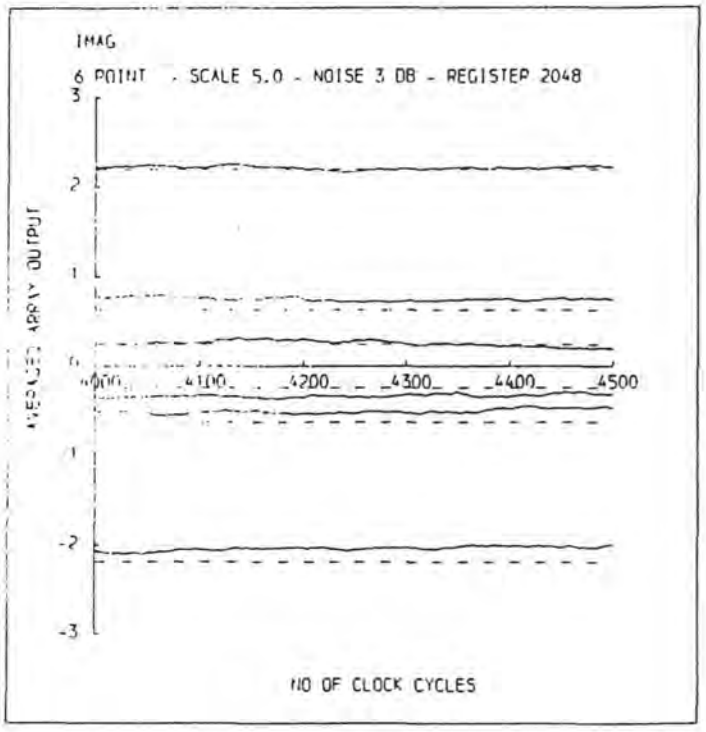


Figure Two (d) : Example array output with 3dB noise input.