AUTOMATA THEORETIC ASPECTS OF TEMPORAL BEHAVIOUR AND COMPUTABILITY IN LOGICAL NEURAL NETWORKS

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

Teresa B. Ludermir

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

November 1990

ABSTRACT

Of the many types of neural networks that have recently emerged this thesis is based on the RAM neuron model [Aleksander-Stonham, 79]. Temporal phenomena are fundamental to the everyday activities of human beings. Because of this, if one is to employ machines which interact with humans, temporal phenomena must be dealt with. The concept of time is at the centre of many fundamental pattern recognition tasks, such as speech recognition and motion detection. In this thesis it is shown that RAM networks are able to store sequential information from input "training" patterns and that they operate quite efficiently on some temporal pattern recognition tasks. Three different systems that deal with temporal problems are presented. The influence of stability in the generalisation and, consequently, the performance of a neural network with respect to a temporal behaviour of the network to solve a given task is discussed. Some ways to control the stability of the network are presented.

The performance of a learning algorithm is measured by looking at the structure achieved through such learning processes and comparing the desired function f to the function computed by the network acting as a classical automaton. It is important to characterise the functions which can be computed by the network in this fixed structure, since if there is no configuration which allows the computation of f, say, then a network cannot learn to compute f. The computability of networks of RAMs and PLNs (Probabilistic Logic Node [Aleksander, 88]) is studied. A new method of recognition based on stored probabilities with PLN networks is suggested. This new method increases the computability power of such networks beyond that of finite state acceptors. It is demonstrated that the computability of a PLN network is identical to the

computability of a probabilistic automaton [Rabin, 63]. This implies that is possible to recognise more than finite state languages with such machines.

The results obtained provide: 1) an insight into the capacity of such networks to deal with temporal pattern recognition tasks; 2) a formal (automata theoretic) relationship between these networks and conventional computation and 3) a more powerful system to recognise temporal patterns.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor, Prof. Aleksander, for his valuable advice, continual encouragement and patience with my limitations. Also, for the freedom to pursue my own ideas and the encouragement to see them through.

I would like to thank my family for the support and encouragement in my dedication to study. Special thanks to Wilson de Oliveira for his understanding, patience and trust during the period of this research.

I would also like to express my special gratitude to Lee Flanagan who was very helpful during these years at Imperial; always with a word of encouragement and friendship, always willing to help.

I would like to thank my colleagues in the neural systems enginnering group for their friendship and the many interesting discussions. Special thanks to Adrian Redgers and Eamon Fulcher for their time in reading this work and making my English readable.

My finance was supported by the Brazilian Research Council CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico).

Table of Contents

Table of Figures

LIST OF SYMBOLS AND ABBREVIATIONS

A	finite state automaton
A_p	probabilistic automaton
$a, b, \ldots$	terminal symbols ($a, b, \ldots \in V_T$)
$D(t)$	desired response or state of the network at instant t
δ	next-state function
ϵ	empty word
F	set of final states ($F \subseteq Q$)
G	regular grammar
G_w	weighted regular grammar
L	language
$L(G)$	language generated by a grammar G
λ	cut-point of a weighted regular language or of a probabilistic automata
MCP	McCulloch and Pitts
N	number of address lines in a node
n	number of nodes in a network
P	set of productions rules
P_w	set of weighted productions rules
PLN	Probabilistic Logic Node
p	weight associated with the application of a weighted production rule
$p(\vec{X} \in L_k/\vec{R})$	probability that the input sequence $\vec{X}$ belongs to the language

	L_k given that the response $\bar{R}$ occurred when $\bar{X}$ was fed into the network
π_0	n-dimensional row vector which contains the initial distribution of a probabilistic automaton
Q	finite set of states of an automaton
q_0	initial state ($q_0 \in Q$)
$q_1, \ldots, q_n$	states of an automaton ($q_1, \ldots, q_n \in Q$)
RAM	Random Access Memory
$\bar{R}$	output sequence or state sequence of a network
$R(t)$	response or state of a network at instant t
SDNN(m,i,f)	Sequential Digital Neural Network with m neurons; each neuron has i terminals connected to the external input and f connected to the feedback input
$S, S_1, \ldots, S_n$	nonterminal symbols ($S, S_1, \ldots, S_n \in V_N$)
S	starting symbol of a sentence
$S_k(t)$	continuous average of $p(\bar{X} \in L_k/\bar{R})$ until the instant of time t
Σ	alphabet (finite set of input symbols)
$T(A)$	set of patterns accepted by A
t	instant of time
V	total alphabet of a grammar
V^*	set of all patterns composed of symbols of V
V_N	nonterminal alphabet of a grammar
V_T	terminal alphabet of a grammar
V_T^*	set of all patterns composed of symbols of V_T

V_T^+	$V_T^* - \epsilon$
$\vec{X}$, $\vec{Y}$	input sequences or input words
$X(t)$	input of the network at instant t
w	a word of a language ($w \in V_T^*$)

CHAPTER 1

INTRODUCTION

The study of artificial neural networks was largely originated in 1943 with the McCulloch and Pitts (MCP) model of the neuron [McCulloch-Pitts, 43]. Much of the current research in the field is still based on their model. There are many advantages in using this model as the basic unit in a neural network, due to the amount of work that has been done on them: many tools for both the development and analysis of such systems have been developed. Nevertheless, the McCulloch and Pitts model (or weighted-sum-and-threshold model) is difficult to realise in hardware because a manipulation of a set of real values (weights) is necessary. Learning algorithms for such a model are very slow and hard to implement. In this thesis a different model is investigated: the logical neuron model or RAM neuron model [Aleksander-Stonham, 79]. The analysis of logical models is still in its infancy and has not been studied as much as the MCP model but the advantages of the model are that firstly, it is very simple to implement in hardware and secondly, learning is not as nearly as slow as the MCP model.

At its origin, much automata theory was motivated by biological and/or psychological investigation. Turing was fascinated by the behaviour of any computer, in the then current sense of a human doing calculations, according to some well specified rules [Turing, 36]. McCulloch and Pitts sought to discern the logical calculus immanent in mental activity by formalising certain basic properties of neurons [McCulloch-Pitts, 43]. Von Neumann sought to combine the work of Turing and McCulloch-Pitts with the emerging study of computers to analyse both brain functions and processes of genetics and reproduction [von Neumann, 51]. In the landmark collection of papers

"Automata Studies" [Shannon-McCarthy, 56], the majority of authors were still concerned with modelling biological and psychological processes, especially neural models. From 1956 on, automata theory came to be more and more the province of applied algebraists and computer theorists, so that the best results of the theory tended to be of mathematical interest, or related to the theory of programming languages. Meanwhile, neural modelling continued slowly but all too often with insufficient mathematical rigour. The field of neural networks has matured and there is more contact between theorists and experimentalists. It seems now appropriate to once again marry automata theory with the study of neural networks. Some questions that arise from such a relationship, which are related to the fact that neural networks are non-universal machines, are:

- Given a training set, how close is the generalisation of the network to the language L to be recognised by the network?
- Which properties must the training set have in order to minimise the difference between the generalisation of the network and the language L to be recognised? That is, given the language L is it always possible to find a training set such that the generalisation is equal to the language? Otherwise, is there a maximal solution?
- What is the computability power of a network? That is, what are the sort of languages that can be recognised by the network?
- What is the complexity of learning in such networks? How many presentations does a network of given components need to learn a certain function?

These are the kinds of question which stimulated the work of this thesis and they were firstly discussed in [Ludermir, 88]. Of course, not all of these question are answered in this work, many are left to future work (section 6.2 of this thesis). In particular, here the class of functions chosen is the recognition

of temporal patterns as applied to the study of logical networks.

The work in this thesis can be seen , in a practical sense, as the study of the ability of logical neural networks to recognise temporal patterns. Three systems using RAM networks are developed to deal with temporal patterns. All the networks have feedback connections once feedback networks are more suitable for temporal pattern recognition. If logical networks are to be used for temporal pattern recognition they need to be able to compute more than finite state languages. Many of the temporal problems cannot be solved by finite states machines. The networks in all three systems are finite state machines. The computability power of the systems is increased by introducing different classifiers combined with the network because the RAM networks, by themselves, cannot deal with temporal patterns.

Another way of increasing the computability power of logical networks is by using PLNs (probabilistic logic nodes) [Aleksander, 88]. A new method of recognition using PLNs is proposed. It is shown that PLN networks with this new method of recognition compute more than finite state languages. They are computationally equivalent to probabilistic automata which makes them more suitable for temporal pattern recognition.

1.1. MOTIVATIONS AND AIM OF THE THESIS

Given the present state of the art, it has been difficult to write programs to solve problems such as sequence prediction, face and scene recognition, etc., using traditional methods on serial digital computers. It is first necessary to find a set of rules (programs or grammars), and, although in some cases it is theoretically possible to find the set of rules, it can take an impractical length of time. In contrast, Neural Networks have been used with considerable success, and in a relatively simpler way, where instead of writting a

program (thereby making the grammar explicit) the net is just "trained" to recognise the patterns. The network acquires knowledge implicitly through the process of training. For example, 20 seconds of exposure to a neurally based system such as WISARD [Aleksander et al., 84] will allow the network to select among a vast number of rules (node functions) in a very short time in order to discriminate between the images in question.

It is not hard to see that in many daily-life pattern recognition tasks the notion of time and sequentiality of actions are fundamental. This is to say that when it comes to understanding memory, learning, and intelligence as manifested in biological systems, the precise temporal relationship among stimuli and between stimuli and responses may be crucially important. Hebb [Hebb, 49] showed an appreciation of this principle in the neuronal model he proposed. He suggested that the efficacy of a modifiable synapse increased whenever the synapse was active in conjunction with activity of the postsynaptic neuron. Thus, Hebb was proposing that learning (i.e., changes in the efficacy of synapses) was a function of correlations between approximately simultaneous pre- and postsynaptic levels of neuronal activity. Models such as Hebb's emphasise the temporal association of signals: each critical event in the sequence leading to learning has a time associated with it and this time plays a fundamental role in the computations that yield changes in the efficacy of synapses.

To deal with temporal pattern recognition tasks, a computational agent must be able to, at least, store (preserve) time information, sequentially in such a way that past computations contribute to future computations. Feedback networks can perform this type of computation. The feedback enhances the temporal process which in the case of word recognition and image recognition leads to selectivity in the network's reaction, while in the case of

language understanding it leads to time-ordered recall. The feedback state stores the sequential information. But feedback RAM networks have only the same computability power of finite state automata, they are finite state machines by definition. Many temporal problems also need the context where the symbol happens and RAM feedback networks are not able to store such contexts. To illustrate the need for context in temporal problems the 'underground journey' example in [Aleksander-Morton, 90] is used.

Consider the map of part of the London Underground shown in figure 1.1. Say that a fully interconnected layer in a network is capable of learning a sequence of states by "being" in a state, and being trained to change to the next state. These states could be the names of underground stations. So if the state of the network is P, it could learn that the next state will be C, in the sense that if the network is clamped to P and released, it will change to C. However, a characteristic of such a layer is that one state can only lead to another specific state. Therefore merely labelling states as stations would lead to confusion, as there are many states that can follow G in the example. Clearly, the network has to store context as a 'state' (e.g. 'I am on the Victoria line at O and I am trying to get C') to determine the next state (which should be 'I am at G changing to the Piccadilly line towards P trying to get to C').

This thesis tackles the problem of the recognition of temporal patterns, initially by using RAM feedback networks associated with different classifiers and, then with PLN networks. The classifiers give the system the extra capacity necessary to work with contextual temporal patterns.

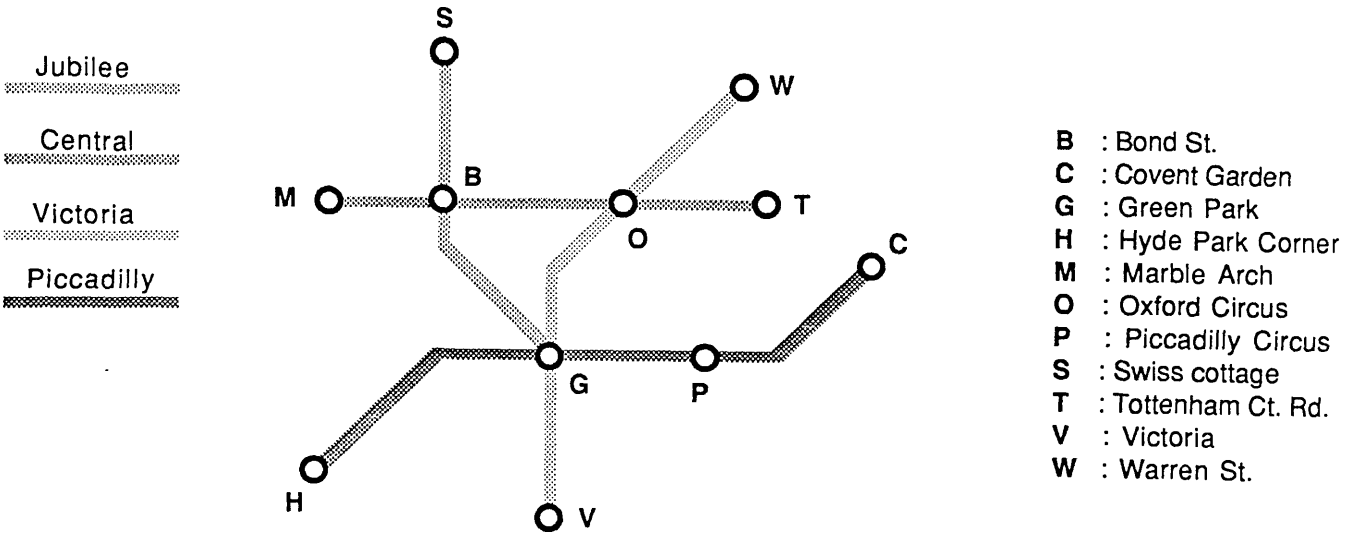


Figure 1.1 A simplified map of part of the London Underground.

1.2. OVERVIEW OF THE SOLUTION AND RESULTS

The temporal problems dealt in this work are the recognition of sequences. Two different types of discrete sequences are considered: i) the sequences of tracking movements for geometric forms and ii) sequences generated by regular languages. The sequences are fed into the network symbol by symbol and the network is trained to predict the next input symbol in the sequence. The systems here were asked to classify sequences into one of two different languages L_1 and L_2. To distinguish more than two languages one can combine many of the systems suggested in this thesis.

The first classifier designed to distinguish the two languages was based on the Information Theory of Shannon [Shannon, 49], that is, the information carried by an input symbol X_i regarding the class membership of an input sequence $\vec{X}$ is directly related to the frequency with which symbol X_i occurs in sequences of both languages L_1 and L_2. Thus the first classifier is based on the probability $p(\vec{X} \in L_k / \vec{R})$ that the input sequence $\vec{X}$ belongs to the language L_k, given that the response $\vec{R}$ occurred when $\vec{X}$ was fed into the network.

$p(\vec{X} \in L_k/\vec{R})$ changes randomly near 1 for sequences $\vec{X}$ in L_k and near 0 for sequences $\vec{X}$ not in L_k. The continuous average $S_k(t)$ of $p(\vec{X} \in L_k/\vec{R})$ is used for considering $p(\vec{X} \in L_k/\vec{R})$ for all symbols in the same sequence $\vec{X}$. $S_k(t)$ tends to 1 for all sequences belonging to the language L_k, while $S_k(t)$ tends to 0 for all sequences not belonging to the language L_k. To facilitate the choice of the parameters of the network, the study of the stability is made. The stability property is responsible for the increase of generalisation and consequently has influence in the recognition of patterns. The influence of the size of the feedback connection on the stability of the network and in pattern recognition is study. Two different ways of controlling the stability are investigated.

The second classifier is also based on the *a posteriori* probability. It is a more powerful classifier in the sense that it is able to process more sequential information. The extra power of this classifier is introduced by the use of more memory, a buffer in this case. Although the buffer gives the system more capability to deal with contexts it also requires more computations to be performed.

The third, and last, classifier is based on automata theory. Automata theory is a well established field and has enabled us to capture some fundamental ideas about the system such as to measure the computability power of logical networks and also to understand the computability power of PLN networks. Unfortunately, this classifier is not very effective; indeed it actually proved to be the worst method, at least in terms of the experimental results. However, the main purpose of this classifier is to measure the capacity of the network to recognise languages as a finite state automaton. If a relationship is found between formal language recognisers and logical networks, the computability power of logical networks can be determined from a knowledge of the computability power of a formal language recogniser. To know the computa-

bility power of a recogniser is the same as to know which functions a recogniser can compute. And it is important to know which functions a network can compute because, obviously, the functions the network cannot compute cannot be learnt.

The computability power of PLN networks is here analysed and the ability of the network to deal with temporal patterns is discussed. The capacity of the PLN networks has not been completely uncovered up to now. The way the recognition algorithms of PLN networks have been designed gives these machines the same computability power as that of RAM networks. RAM networks are finite states machines and can only compute regular languages. The advantage of the PLN model which is being exploited is in the training of the network. PLN networks are easier to train because there is no pre-existing structure (depending on the initial value of the memory of the nodes before training, arbitrary confluences of states exist) in the state space before training [Kan-Aleksander, 89]. The nodes of the network are initialised with the value of 'undefined' before training. This initial value signifies a guess to the correct answer is required and consequently this 'undefined' value separates the untrained memory position from the trained ones.

A new recognition algorithm is here proposed that uses all of the computability power of the PLN network. This recognition algorithm makes use of the probabilistic information stored in the memory of the node. It is proved that the computability power of a PLN network is the same as of a probabilistic automaton. Probabilistic automata can recognise more than finite state languages [Rabin, 63]. There are context-free, context-sensitive and recursive languages that can be recognised by probabilistic automata and thus they are more suitable for dealing with symbol sequences than finite automata. The proof of the equivalence between PLN networks and probabilistic automata is

divided in two parts. In the first part an algorithm is given to transform any weighted regular grammar [Salomaa, 69] into a PLN network. With this algorithm it is possible to design a network to recognise a specific weighted regular language. This algorithm also provides a way of implementing probabilistic automata in non-deterministic devices. Changes can be made in the PLN network through training in order to recognise a different language from the one the weighted regular grammar generates. The second part of the proof demonstrates that every set of patterns recognised by a PLN network can be generated by some weighted regular language. It will be also possible to determine the generalisation of a network and the functions a network is able to compute using the algorithm designed in the second part of this proof - which may be a useful contribution in itself.

1.3. OUTLINE OF THE THESIS

This thesis contains six chapters. Here, in chapter one, the work carried out is stated, and the organisation of the thesis is outlined. Chapter two presents a brief review of research on artificial neural networks covering the work both in the weighted-sum-and-threshold models and in the logical neuron models. The current state of research on processing temporal patterns using neural networks will be reviewed, and previous work in this area with logical neural networks is discussed.

Chapter 3 is concerned with Automata Theory, and more specifically with probabilistic automata theory. An overview of the research done on probabilistic automata is given and all definitions used in this thesis from this field are explained. Some neural networks designed for formal language recognition are described.

Chapter 4 contains all the experimental results of the thesis. It describes the nature of the networks and the data used in the experiments. It explains why feedback networks are more suitable for temporal pattern recognition than feed-forward networks. A probabilistic classifier is suggested with which recognise input sequences. The effectiveness of the system is illustrated by the experimental results. The influence of the stability property in the ability of the network to solve a specific task is discussed. Two more classifiers, the buffer and final state classifiers, are suggested and further experimental results are shown. A comparison of the three classifiers is then made.

Chapter 5 is dedicated to the study of the computability of logical networks. A new recognition method using PLN networks is proposed. The equivalence between the computability of a PLN network and a probabilistic automaton is demonstrated. An algorithm that converts any weighted regular language into a neural network is designed and an example with a particular language is presented. Another algorithm to generate a weighted regular grammar for the patterns recognised by a given network is put forward and followed by an example. Different ways of training the network generated from the grammar are suggested.

Chapter 6 reviews the results of the work as a whole and the contributions of the thesis. It discusses the nature of the main results achieved and indicates those areas in which it is considered likely that further research will be valuable.

CHAPTER 2

NEURAL NETWORK MODELS AND TEMPORAL PATTERN RECOGNITION

2.1. INTRODUCTION

As part of the recent revival of interest in artificial neural networks, many new approaches have been produced. Before making a contribution to the field, the developments in the area should be stated. The study of Neural Networks is not entirely new, indeed its past stretches back beyond that of conventional computing. What is new, however, is a concern with well-founded analysis and a deepening understanding. In this chapter there is a review of the most important work done on the two major models of neural networks: the weighted-sum-and-threshold model and the logical model.

Much of the enthusiasm for investigating neural networks lies in their potential in applications for which solutions are unsatisfactory when attempted using conventional computing. Temporal pattern recognition is one of the areas in which neural network solutions are being studied. Many problems in this field, such as in the field of speech recognition, are considered as being difficult when approached with conventional techniques. Here, descriptions of neural solutions for temporal pattern recognition problems are given.

2.2. WEIGHTED-SUM-AND-THRESHOLD MODELS

The neuron was first modelled in 1943 by McCulloch and Pitts [McCulloch-Pitts, 43]. This was a simplification of what was known of a real neurons. They noted the following properties of real neurons: that they had many inputs but a single output which could branch towards other neurons;

that some of the inputs were excitatory (had positive weights) and some were inhibitory (had negative weights); and that neurons would only fire if the excitation on its inputs exceeded the inhibition by a certain threshold T. They simplified the model by assuming that the neuron acted in a synchronous way, so that on some absolute time scale the inputs at time t to any neuron determine the output at time $t+1$. Despite these simplifications, it is still this model that is the basis of neural networks in contemporary neural computing.

If networks of such formal neurons that contains feedback connections are considered then it is observed that there is a set of inputs, a set of internal states and an output set related to every network. Recalling the critical time scale assumption on the neurons, the state and input of the whole network at any one time can establish the state of the whole network at the next time cycle as is conventional in finite-state machines. To determine the firing of a neuron it is necessary to know the previous firing pattern of its inputs, and this is induced partly by the overall input to the network and, by the state of the network. It is clear then from the above discussion that any network composed of McCulloch and Pitts neurons (henceforth referred to as MCP neurons) is an example of what is called a finite state automaton [Arbib, 87].

McCulloch and Pitts suggested an implementation of their neuron by means of summing amplifiers, where variable input weights w (which can take values in the range -1 and +1) play a role analogous to that of synapses in natural neurons - giving the neuron its adaptability. The neuron takes firing signals at all its synapses into account by summing their effects, both excitatory and inhibitory and thereby deciding whether it should or should not fire. The neuron fires if the total sum exceeds the threshold T. In the MCP model it is assumed that the firing at the axon of a neuron may be represented by the number 1 and no firing at the axon by the number 0. When it is not known

what this number is, the state of the axon of the neuron is given the label x. The effect on a neuron of any particular synapse is the product xw. The mathematical form of the firing rule for a MCP neuron is:

The neuron fires if the following inequality is true:

$$x_1w_1+x_2w_2+\cdots+x_nw_n>T$$

or, in more compact form:

$$\sum_{j=1}^{n} x_jw_j>T$$

where n is the number of synapses of the neuron, w_j is the weight associated with input x_j and T is the threshold.

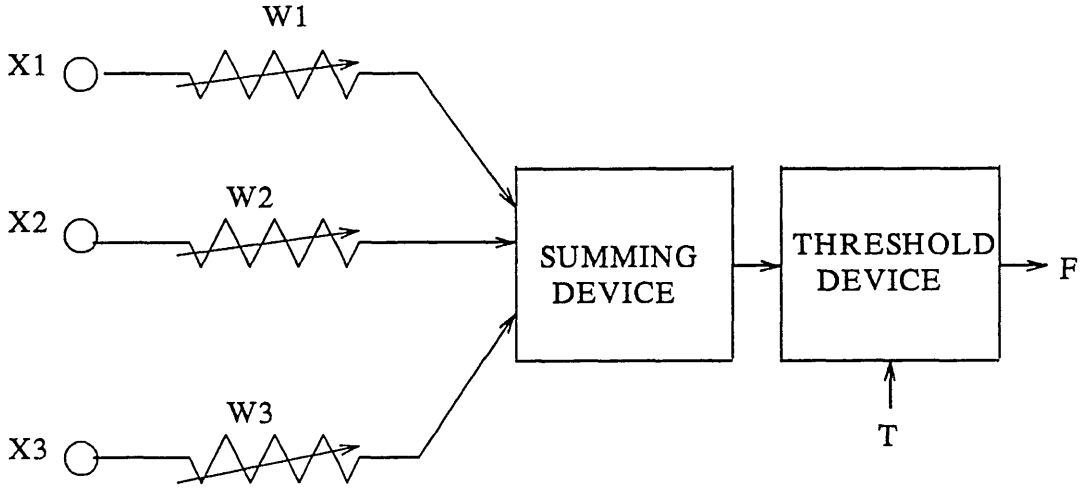


Figure 2.1 The McCulloch and Pitts model.

Much discussion of neural networks focuses on the methods for training these nodes to perform a particular function. Many training techniques used until now are based on what Hebb [Hebb, 49] first suggested in 1949.

> when an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased [p.50]

He suggested that cells that are frequently active should have an increased chance of becoming active again adding the concept of learning to the MCP

model by suggesting that synapses are the site of biological learning. Hebb did not develop a mathematical rule for his learning suggestion. It was up to other researchers to produce more explicit statements of Hebb's idea.

Another important learning rule was first suggested by Widrow [Widrow, 62]. This is now widely used and it is known as the Widrow-Hoff rule or the delta rule. The Widrow-Hoff learning rule is local, that is, all that is required by a node is the input, the output and what the output should have been. The rule works by calculating an error signal between the desired output and what the sum (of synapse and weights) computed. The weights are slowly changed to reduce the error until no more error exists. Back-propagation, as described by Rumelhart et al [Rumelhart et. al, 86] is a generalisation of the Widrow-Hoff rule applied when the output error of neurons buried inside a net cannot be directly assessed.

Another important line of development was made by Rosenblatt with his "Perceptrons" [Rosenblatt, 58]. The basic element of a perceptron is a MCP node with some additional, fixed, preprocessing. This is shown in figure 2.2. The units A_1, A_2, ..., A_p have the function of extracting specific features from some input image, and are called association units. Perceptrons were presented as pattern recognition devices. They are trained using the delta-rule previously described. Block [Block, 62] proved that a perceptron with no more than one layer could be trained in finite time (the convergence property). If any learning rule is to be usable it must certainly have this convergence property.

In 1969 Minsky and Papert [Minsky-Papert, 69] drew attention to some tasks which perceptrons could not perform. Perceptrons, for example, cannot detect 'connectedness' and 'parity'. These are examples of 'hard learning problems'. Hard learning problems form a very large class of functions and

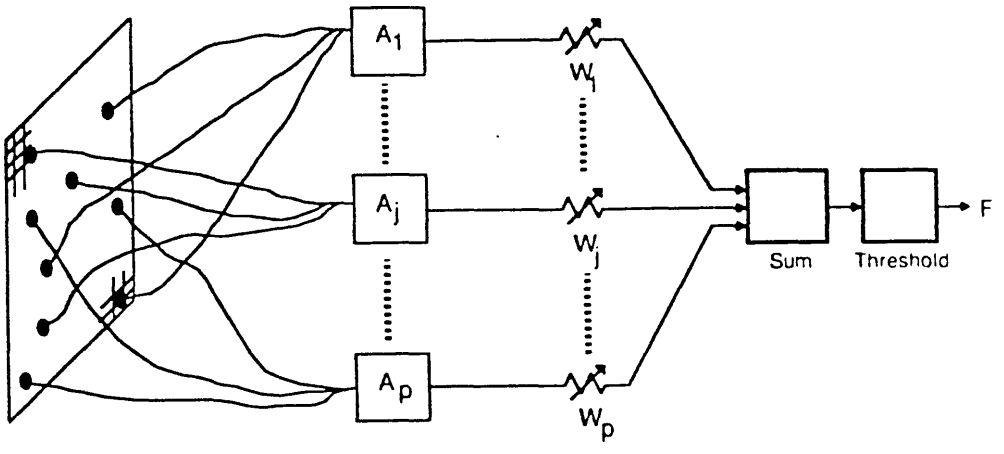


Figure 2.2 A Perceptron

cannot be neglected. Although perceptrons cannot learn such problems, multi-layer networks can, but it is difficult to find the right learning algorithm to train a multi-layer network to solve hard learning problems. Additionally, in their book, Minsky and Papert claimed that if a network of perceptron nodes can solve a particular problem, the solution found will not scale up. It is also important to realise that hard learning problems can sometimes be easily solvable with conventional computing.

Rosenblatt was himself aware of the limitations of single layer perceptrons. However, if the nodes are arranged in multiple layers with hidden nodes (nodes that are not accessible from the external input and output) it is possible to show that any function can be achieved. But a generalisation is needed of the training algorithm since with multi-layer networks, the delta rule, as defined before, does not apply. Rosenblatt then proposed to propagate the error backward from the external output layer to the hidden nodes. However the computational complexity of this method proposed by Rosenblatt, to multi-layer perceptrons, increases exponentially with the size of the

network. Thus, when one tries to enlarge the scale of a perceptron-like machine the numbers of steps of computations are increased exponentially with the size of the problem. This was another of Minsky and Papert's criticism of perceptrons. A consequence of the work of Minsky and Papert and the success of Artificial Intelligence led to a general decline of interest for neural networks in the seventies.

Despite these difficulties, many researchers continued to explore Artificial Neural Networks such as Igor Aleksander (logic neuron model) at Imperial College, Kunihiko Fukushima (cognitrons and neocognitrons) at NHK Science and Technical Laboratories, Steven Grossberg (self-adaptive systems) at Boston University, Teuvo Kohonen (self-organised associative memory) at Helsinki University among others. Their results were, however, scattered among many journals in different areas, making their work largely unknown to most researchers in Artificial Intelligence until recent years.

In 1982 Hopfield [Hopfield, 82] published a paper which attracted attention to the associative properties of a class of neural networks. This paper was responsible for a revival of interest in the analysis of neural networks. According to Hopfield, the possible activations of a network can be viewed as its state space, its current activation is its position in state space. The system moves over time until it reaches a stable limit point. Hopfield defined an 'energy function' for the current state of the system and showed that energy steadily decreases and eventually relaxes into a stable state. A problem, however, is that the energy may settle into a local minimum (one solution of many) instead of the global minimum (the optimal solution). It was Hinton, though, who suggested a way of overcoming these problems through what he called 'the Boltzmann machine' [Hinton et. al, 84], which is based on ideas originating in statistical mechanics and thermodynamics. Noise is added to the

system (the equivalent of heat in the Boltzmann system) to avoid false global-minima (local minimum).

However, Hopfield's analysis still did not tackle the problems of hard learning which was the main criticism of Minsky and Papert. A solution for such problems was necessary to make neural networks carry out useful computations. After the definition of the Boltzmann machine Rumelhart, Hinton and Willians developed and formalised the Error Back-Propagation or Generalised Delta Rule which deals with the hard learning problems [Rumelhart at al., 86]. Error back-propagation allows the training of hidden nodes, but requires feed-forward topology.

There are other training methods to deal with hidden nodes and hard learning problems in MCP networks. Some of these training methods are unsupervised. In this mode, the network discovers hidden patterns in input data which even the designer of the net may not be aware of. These unsupervised learning methods have an important role to play in systems such as speech recognisers. The main work on unsupervised learning has been conducted by Fukushima [Fukushima, 73], Grossberg [Grossberg, 76] and Kohonen [Kohonen, 89]. Such training methods are largely outside the scope of this thesis.

2.3. LOGICAL NEURON MODELS

Neural network models based on weighted-sum-and-threshold have been extensively studied in neural computing as was suggested in the last section. The neural computing model used in this thesis is based on a different kind of artificial neuron called the "logical neuron model" or the "weightless" neuron model. The logical model is based on the simple operations of a look-up table which is best implemented by random access memory (RAM) and where

the knowledge is directly "stored" in the memory (via "look-up tables") of the nodes during learning. Some advantages of this model are: (1) systems may be built using conventional digital circuits, without the need to develop special VLSI devices; (2) learning is not unreasonably slow and (3) error-correction requires only a global success signal. A definition of a RAM neural network is given below:

Definition 2.1 - A RAM Neural Network is an arrangement of a finite number of neurons in any number of layers, in which the neurons are RAM (Random Access Memory) nodes. The RAM node is represented in the figure 2.3 below:

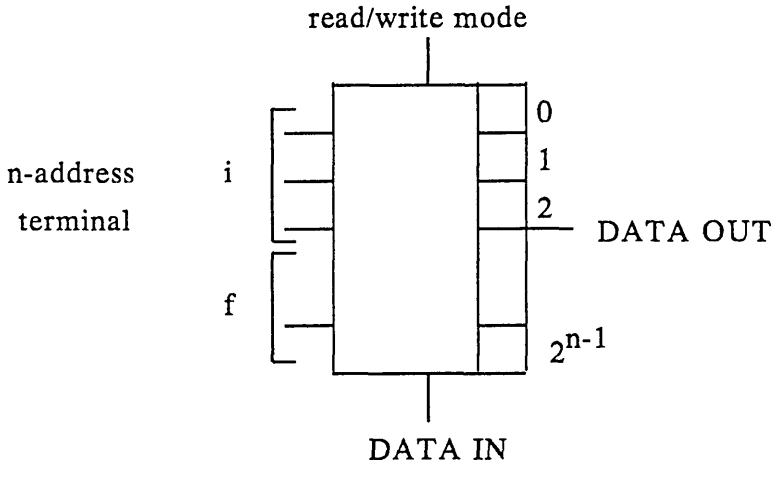


Figure 2.3 RAM node

The input may represent external input or the output of neurons from another layer or a feedback input. The data out may be 0's or 1's. The set of connections is fixed and there are no weights in such nets. Instead the function performed by the neuron is determined by the contents of the RAM - its output is the value accessed by the activated memory location. There are 2^{2^N} different functions which can be performed on N address lines and these correspond exactly to the 2^N states that the RAM can be in, that is a single RAM can

compute any function of its inputs. Figure 2.4 below is one example of a RAM neural network.

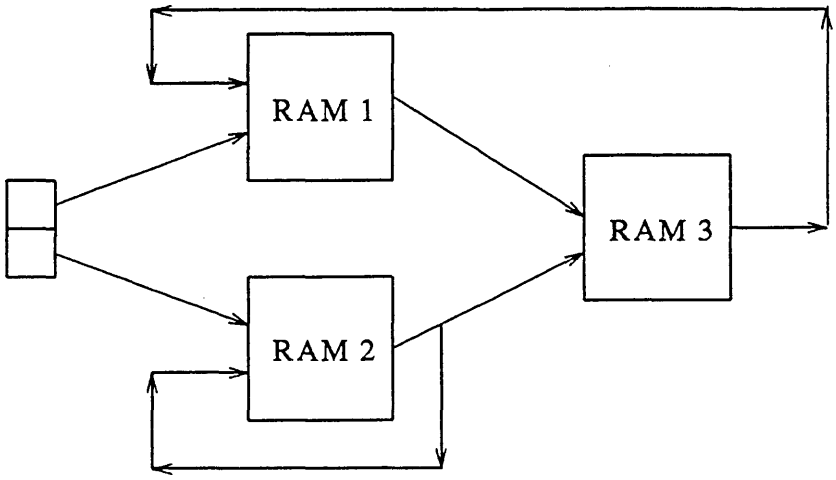


Figure 2.4 An example of a RAM neural network.

Learning in a RAM node takes place simply by writing into it, which is much simpler than the adjustment of weights as in a weighted-sum-and-threshold network. The RAM node, as defined above, can compute all binary functions of its input while the weighted-sum-and-threshold nodes, defined in section 2.2, can only compute linearly separable function of its input. There is no generalisation in the RAM node itself (the node must store the appropriate response for every possible input), but there is generalisation in networks composed of RAM nodes [Aleksander, 83]. Generalisation in logical networks is affected first by the diversity of the patterns in the training set, that is, the more diverse the patterns in the training set, the greater will be the number of subpatterns seen by each RAM, resulting in a larger generalisation set. Secondly, the connection of RAMs to common features in the training set reduces the generalisation set. Three types of misclassification can come from the generalisation of a network:

1) rejection by double: intersection of two or more generalisation sets;

2) unknown rejection: a pattern $\vec{X}$ belonging to one class falls outside the generalisation set of this class; and

3) error: a pattern $\vec{X}$ belonging to one class falls within the generalisation set of another class.

The logical node is based on the n-tuple sampling machines of Bledsoe and Browning [Bledsoe-Browning, 59], where the n inputs to the node form the n-tuple which is used to address node memory. While Aleksander took a great deal of interest in adaptive learning networks using n-tuple sampling machines, Kauffman's interest was in gene regulation [Kauffman, 69]. Kauffman examined the model of genes by random binary automata with two inputs.

Aleksander suggested a universal logic circuit as the node of a learning network [Aleksander, 66] and [Aleksander, 71]. He introduced the SLAM (Stored Logic Adaptive Microcircuit) node, which was produced specially for research purpose before the availability of integrated circuit memories, and the RAM node as in definition 2.1. Cheung investigated the use of RAM feed-forward networks in hand-written numerals [Cheung, 73]. Stonham also used RAM feed-forward networks as feature extractors in the classification of mass-spectra [Stonham, 74]. Feed-forward RAM networks were also used by Nappey in the input pattern recogniser of a reading aid for the blind [Nappey, 77]. In the applications mentioned above, a set of RAM networks is normally used as discriminators (or feature extractors) with one discriminator per response class. Reeves and Dawson used a set of RAM networks as the adaptive controller of a TV camera in a scene analysis system ([Reeves, 74] and [Dawson, 76]). Reeves also worked with moving edge detection using a feedback RAM network. This part of his work is described in section 2.4 of this chapter where the work with feedback RAM networks is covered.

By the late 1970's, the construction of a large network able to deal with high quality images became economical as RAM elements were much cheaper and of a large capacity. In 1979 the WISARD (Wilkie, Stonham & Aleksander's Recognition Device) [Aleksander-Stonham, 79] was designed and the prototype was completed in 1981. The WISARD is based on RAM nodes with n ($2 \leq n \leq 16$) inputs. During the operation of the system, an incoming image is stored and digitised. Each pixel is represented by a single bit. Each of the n inputs of a RAM node takes a binary value from a pixel using a random, but fixed mapping. It is assumed there are enough RAMs to map into every pixel. The binary values taken from pixels form n-tuples which are the addresses specifying where the read/write operations are performed. A layer of RAMs is called a discriminator and represents a class of objects to be recognised. A WISARD system consists of a number of discriminators and a decision unit which calculates the responses and the confidence of the discriminators during a recognition process.

The recognition activity is built up by a process of training on examples of the object to be recognised. An object is presented through a camera together with its desired classification. All the storage locations of discriminators are set to 0 before the training process. The classification selects a particular discriminator to be trained. 1's are stored in all the locations addressed by n-tuples taken from the input image but only within the selected discriminator. This has the effect of causing that particular discriminator to produce a logical 1 at all its data output terminals if the training image is presented again. If the input image is slightly changed then not all the RAMs within the selected discriminator will respond with 1. The percentage of RAMs in a discriminator responding with a 1 is called the 'response' of that discriminator. The responses are a function of the overlap of the input patterns and the

trained patterns. In a recognition process, the responses of discriminators are fed into a simple calculator which first identifies which of the discriminators has the strongest response and outputs the class number associated with that discriminator. It also provides the actual response of the discriminator and the next highest one. This difference provides a measure of confidence with which the decision is made. The WISARD was patented and produced commercially in 1984.

RAM networks have been used for many tasks and here only a selection of them are presented. Fairhurst and Mattoso-Maia proposed a recognition system consisting of two layers of RAM nodes which was capable of storage reduction in the recognition of alphanumeric characters [Fairhust-Maia, 83]. Aleksander and Wilson demonstrated that RAM networks when trained to be edge-detectors can perform at least as well as Sobel and Laplace transforms on a binary image, and are faster and less sensitive to noise distortion [Aleksander-Wilson, 85]. Allinson and Johnson have used unsupervised learning in a n-tuple recogniser to classify video images at video speed [Allinson-Johnson, 89]. Tattersall, Foster and Linford used a n-tuple single layer network (that is, a WISARD architecture) to do speaker independent recognition of utterances of the letters of the alphabet and to do text to speech synthesis [Tattersall et. al, 89]. Their learning rule is different from the one used with the WISARD, in their case the training rule is based in the difference (error) between the output of the network and the desired output. This error vector is used to modify the values of the current addressed RAM locations so that next time the same input vector is applied, the output is nearer to the desired output.

Recently, the problem of moving edge detection has been used again as an example of a problem solvable with logical networks [Vidal, 88]. Vidal

also analysed the performance of logical networks in the problem of localisation and tracking patterns of some minimum size in the input data. The RAM nodes of Vidal's network are organised in pyramid (tree) structures.

The introduction of a probabilistic element into the logic node was proposed by Aleksander [Aleksander, 88]. He called the node with this probabilistic element a probabilistic logic node (PLN). The main feature of the PLN model is the unknown state, u,which responds with a randomly generated output for inputs on which it has not been trained. Below is given a definition of a PLN node.

Definition 2.2 - A PLN Neural Network is an arrangement of a finite number of neurons in any number of layers, in which the neurons are PLN (Probabilistic Logic Node) nodes. A PLN node differs from a RAM node in the sense that a q-bit number (rather than a single bit) is now stored at the addressed memory location. The content of this location is turned into the probability of firing (i.e. generating a 1) at the overall output of the node. Say that q is 3, then the numbers 0 to 7 can be stored in each location of the PLN. One way of regarding the actual number stored may be as a direct representation of the firing probability by treating the number as a fraction of 7. So a stored 2 would cause the output to fire with a probability of 2/7, and so on. The difference between a PLN neural network and a RAM network is then that (a) memory locations may store q-bit numbers (where $q>1$) and (b) the stored value represents the probability of outputting a 1.

PLN networks have some advantages relative to RAM networks. First, because there are no pre-existing structures in the state space of PLN nets, they are easier to train [Kan-Aleksander, 89]. There are a number of training strategies for PLN networks such as the ones in [Aleksander-Morton, 90], [Myers, 88] and [Al-Alawi-Stonham, 89]. Second, while RAM nets are not

very sensitive to small differences in input patterns, PLN nets can be made very sensitive if they are organised into a pyramid of PLNs [Aleksander-Morton, 90]. Thirdly, there are various sources of noise in the activity of natural neurons in neural networks [Taylor, 87]; and with the stochastic activity of PLNs a slightly more realistic modelling of neural activity is achieved than with RAMs. Forthly, by experimental results, it is known that when solving the same problem with PLN nets and RAM nets, it is possible in many cases to save states when using PLN nets. This means that smaller number of nodes are necessary when using PLN networks than when using RAM networks.

In order to make PLN less susceptible to noise and more able to generalise Aleksander [Aleksander, 90] suggested an extension of the PLN, the G-RAM, which once trained, spreads stored information to those neighbouring locations which still are in the unknown state u.

Another extension of PLN was suggested by Gorse and Taylor [Gorse-Taylor, 88]. They developed a model of a noise neuron which incorporates and formalises many known properties of living neurons. They called their node model a p-RAM. The p-RAM in its simplest form is a lookup table in which each address stores a value $q \in [0,1]$. This q value is the probability of firing (i.e. generating a 1) at the overall output of the node. The main difference between p-RAM and PLN is that p-RAMs allow continuous values to be stored in the memory of the nodes whilst PLNs allow only discrete values. A hardware implementation of a 2-node network of 2-input p-RAMs has been constructed successfully [Clarkson et. al, 89].

2.4. TEMPORAL PATTERN RECOGNITION

Temporal phenomena are fundamental to the everyday activities of human beings. As a result, time is at the centre of many pattern recognition tasks, such as speech recognition, motion detection, and signature verification. Conventional computing cannot, at the moment, satisfactorily deal with many of these pattern recognition tasks where time is a parameter. Connectionist learning algorithms to date have only marginally been considered for applications to time-varying input patterns. Although few good results have been achieved using Neural Networks in difficult temporal problems, such as speech recognition, it is widely accepted that the way they learn, in many aspects similar to human brains, should make them more adequate to work with such problems than conventional machines. Neural Networks will have limited applications if they cannot be shown capable of reasoning in a temporal sense.

Several neural models have been developed to deal with temporal patterns such as TRACE [McClelland-Elman, 86] and focused networks [Mozer, 88], and others, such as the neocognitron [Fukushima, 90], are being adapted to the temporal domain. There are also models based on logical networks which have been shown to work with problems like sequence recognition [Ludermir, 90b]. All these models, nonetheless, need improvement in order to operate efficiently. A short description of some of these follows:

TRACE is a model which assembles phonemes into meaningful phrases. To put together the whole sentence from parts of a signal that is spread out in time, a buffer was used. Such a buffer holds the n most recent signals of the input sequence and can be implemented using a shift register. The buffer turns a temporal recognition problem into a spatial recognition problem in which all relevant information for making a response is simultaneously

available. Because connectionist models are relatively good at spatial recognition problems, this approach works with some success. There are many other models for dealing with temporal pattern recognition which use a buffer. Some drawbacks of the buffer model are: 1) the buffer must be sufficient in size to accommodate the longest possible input sequence; 2) each element of the buffer must be connected to a higher layer of the network. In consequence a large number of training examples must be used, else the network will not generalise well. to obtain nonlinear interactions across time.

Simple temporal (or sequential) behaviour, where information only needs to be stored over time, in a single-layer logical network is achieved simply by making the networks recurrent, that is, some input terminals of the nodes are fed from output terminals with some delays (feedback information) [Ludermir, 89a]. The idea of feeding back the output information to the input with logical networks was introduced by Aleksander and Mamdani [Aleksander-Mamdani, 68]. In their paper Aleksander and Mamdani already mentioned that feedback connections might bring improvement in the performance of such learning networks. Later, Aleksander and Mamdani [Aleksander-Mamdani, 70] applied feedback networks in sequential tasks that are universal. They also discussed, in this paper, that even though universal networks with feedback have a behaviour which is easily understood, their analysis is not trivial.

Feedback networks overcome many limitations of perceptron-like machines. The reason perceptron-like machines cannot decide, for instance, whether or not an input pattern is disconnected is because each individual element has only a restricted knowledge of the whole input pattern; whereas feedback networks (sequential machines) if properly trained (or designed) are capable of computing global properties since the elements are allowed to com-

municate with one another and the feedback input of the network spreads the knowledge of each individual element out to the whole network. Thus, it is obvious that feedback networks do have interesting properties. There are temporal problems which, besides needing information being stored over time, also need the position (context) where the symbol occurs (as explained in section 1.1 of this thesis) in the patterns, and RAM feedback networks are not able to store such a context. One solution to this problem of saving context is to use PLN networks, as is explained in chapter 5 of this thesis.

Some work has been done on the application of feedback logical networks to the problems of temporal pattern recognition. Fairhurst describes a system in which 'OR feedback networks', with random connections, can be made sensitive to sequences after being exposed to a training process [Fairhurst, 73]. In these networks the logical OR of the present input pattern and the previous output pattern is formed, and the resulting pattern is fed to the nodes, which then produces the present output pattern. The network was trained to reproduce the input sequence, that is, the present output is trained to be equal to the present input. In a simulation of a small network of this type, he has observed clustering of the output state cycles as the result of training on two classes of input pattern.

Reeves has also used a network incorporating feedback in an attempt to recognise the sequences of patterns that are generated when a viewing window follows round the edge of a pattern [Reeves, 74]. The nodes can be trained to make the viewing window follow round the edge of a figure. Once the system has been trained on a few shapes, such as a square and a triangle, it can generalise and follow the contours of others shapes, such as circles. Reeves' networks had direct feedback, that is, the present input and previous output are fed directly to the nodes. Reeves never achieved the desired classification

which was in itself an important result, for it pointed out the need for a more fundamental approach to the problem. The unsatisfactory results achieved by Reeves are due to the configuration he selected, which was, in effect, a two layer RAM network with feedback loops. Multi-layer networks are known to be difficult to train and techniques like back-propagation were not used at the time.

Tollyfield investigated the properties of networks with feedback [Tollyfield, 75]. He was particularly concerned with the training algorithms required to enable this type of network to solve recognition problems. He was looking for a training algorithm which would be generally applicable to a wide range of tasks. What emerged, however, was that the deriving of training algorithms is a far more complex problem that was at first envisaged, that a unified approach would be almost impossible. He realised that the inherent behaviour of sequential network restricts the selection of training strategies. Networks with feedback loops possess memory of previous events and so one needs to consider the feedback information during training. It is more difficult to control cyclic (temporal) activities than static (spatial) responses. In the case of temporal patterns the design of training strategies is even more complex because the information (which is to be absorbed by the network) concerning the class membership of a pattern is spread out in time.

Fernandes applied feedback networks to the problem of temporal recognition [Fernandes, 77]. Two particular cases of segmented temporal patterns were considered in his work: the recognition of prototype sequences with errors and the recognition of sequences of tracking movements in a scene analysis problem. The technique proposed involved not only the use of sequential networks but also the use of combinational networks and the use of conventional storage systems. He showed that sequential networks are

inherently stable sequential structures, and with the recognition of prototype sequences with errors there was no need to train the network. He proposed a technique where input sequences are transformed by a sequential RAM network into points in an one-dimensional Euclidean space. The sequential network was shown to reduce the complexity of the discrimination process.

2.5. SUMMARY OF THE CHAPTER

This chapter has presented some of the work done in Artificial Neural Networks since the first definition of an artificial neuron. Some of the history of neural networks has been reviewed: the reasons for a 'dark age' in the seventies and the reasons for the revival of the current interest. Also some of the ways in which neural networks have been trained and used were outline.

The logical model was described, including the definitions of the RAM and the PLN nodes. A comparison between the two nodes was delineated, where the advantages of each node were reported. The WISARD system, based on the RAM node, was explained. Some other tasks using the RAM node were covered.

The last section of the chapter related some of the difficulties of designing neural networks to solve temporal problems. The work connected with the topic of this thesis, using the logical node, was explained.

This chapter was intended to prepare the way for the discussions which follow for the rest of this thesis, by presenting the advantages and disadvantages of the different neural models. It gave some of the reasons why the logical node was chosen to be used in the system described in chapter 4.

CHAPTER 3

AUTOMATA THEORY

3.1. INTRODUCTION

To make the thesis self-contained all the concepts used from Automata Theory are defined in this chapter. The definitions are followed by examples which are going to be used again in chapter 5. Although the literature on deterministic and non-deterministic automata is quite substantial, not quite as much work has been done on probabilistic automata [Rabin, 63]. As a background to the current chapter and chapter 5 of this thesis, a brief overview of the work done in probabilistic automata is given in the fourth section of this chapter.

In the last section, some applications of neural networks designed to formal language recognition are described. Such networks have the same computability power of finite state automata. Some of these systems are able to recognise very simple context-free language with the aid of extra memory. In chapter 5, it will be shown that logical networks can compute more than finite state automata without the need of extra memory.

3.2. FINITE MACHINES

Finite-state languages (or regular languages) are generated by the simplest of the formal grammars, that is regular grammars, and are recognised by finite state automata. In this section a regular grammar and a finite state automaton will be defined. Each definition is followed by an example.

Before giving the definitions of a regular grammar and a finite state automaton some notation need to be introduced. If V_T is the terminal alphabet, the V_T^* denotes the set of all patterns composed of symbols of V_T, including the empty pattern ϵ. V_T^+ denotes the set V_T^* - ϵ, where ϵ is the empty word.

Synonyms for pattern are *string*, *sequences* and *word* and throughout this thesis two different notations for patterns $\bar{X}$ and w are used.

Definition 3.1 - A regular grammar G is a 4-tuple $G=(V_N,V_T,P,S)$ in which:

1. V_N and V_T are the set of nonterminal and terminal symbols of G, respectively. V_N is supposed to have k elements. The union of V_N and V_T constitutes the total alphabet V of G and $V_N \cap V_T = \varnothing$.
2. P is a finite set of rewrite rules or productions denoted by $S_1 \rightarrow wS_2$ or $S_1 \rightarrow w$ where S_1, $S_2 \in V_N$ and w is a pattern of terminal symbols ($w \in V_T^*$).
3. $S \in V_N$ is the starting symbol of a sentence.

It is assumed that the nonterminal variables are ordered S_1, S_2,...,S_k and if the production rule has the form $S_i \rightarrow wS_j$ where $i \geq j$ it is a recursive production rule. Note that this does not impose any restrictions in the regular grammar yet it simplifies the structure of the algorithms proposed in this thesis.

In order to define the language a grammar generates, the relations $=>$ and $\overset{*}{=>}$ between strings in V^* need to be defined. If $\alpha \rightarrow \beta$ is a production of P and γ and ζ are any strings in V^*, then $\gamma\alpha\zeta => \gamma\beta\zeta$. Suppose that α_1, α_2, ..., α_m are strings in V^*, and $\alpha_1 => \alpha_2$, $\alpha_2 => \alpha_3$, ..., $\alpha_{m-1} => \alpha_m$. Then $\alpha_1 \overset{*}{=>} \alpha_m$.

The language generated by a grammar G is $L(G)= \{ x \mid x \in V_T^*$ such that $S \overset{*}{=>} x \}$.

Below an example of a regular grammar is given.

$G=(V_N,V_T,P,S)$ where $V_N=\{S,S_1\}$, $V_T=\{a,b\}$ and

P: (1) $S \rightarrow aS_1$

(2) $S_1 \rightarrow aS_1$

(3) $S_1 \rightarrow b$

The language $L(G)$ generated by G is $L(G)=\{x \mid x=a^nb$ with $n=1,2,\ldots\}$

Note that, as defined here, ϵ can be in no regular language. If L had a finite description then $L_1=L \cup \{\epsilon\}$ would likewise have a finite description of L_1. The definition of regular grammar will be extend to allow production rules of the form $S \rightarrow \epsilon$, where S is the start symbol, provided that S does not appear on the right-hand side of any production rule. In this case, it is clear that the production rule $S \rightarrow \epsilon$ can only be used as the first step in a derivation.

Definition 3.2 - A finite state automaton A is a 5-tuple $A=(\Sigma,Q,\delta,q_0,F)$ where Σ is a finite set of input symbols (alphabet), Q is a finite set of states, δ is a mapping of $Q \times \Sigma$ into Q (next-state function), $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states. The set of patterns accepted by A is defined as $T(A)=\{x \mid \delta(q_0,x) \in F\}$.

There exists a one-to-one relationship between the languages generated by regular grammars and the sets accepted by finite state automaton which can be expressed by the following theorems (the proofs for these theorems are omitted here as they can be found in any text book of Automata Theory, such as [Hopcroft-Ullman, 79]).

Theorem 3.1. *Let $G=(V_N,V_T,P,S)$ be a regular grammar. Then there exists a finite state automaton $A=(\Sigma,Q,\delta,q_0,F)$ with $T(A)=L(G)$.*

Theorem 3.2. *Given a finite state automaton $A=(\Sigma,Q,\delta,q_0,F)$, there exists a regular grammar $G=(V_N,V_T,P,S)$, such that $L(G)=T(A)$.*

An example of a finite state automaton is:

$A=(\Sigma,Q,\delta,q_0,F)$ where $\Sigma=\{a,b\}$, $Q=\{q_0,q_1,q_2\}$, $F=\{q_1\}$ and

δ: $\delta(q_0,a)=q_0$; $\delta(q_0,b)=q_1$; $\delta(q_1,a)=q_2$; $\delta(q_1,b)=q_2$; $\delta(q_2,a)=q_2$; $\delta(q_2,b)=q_2$;

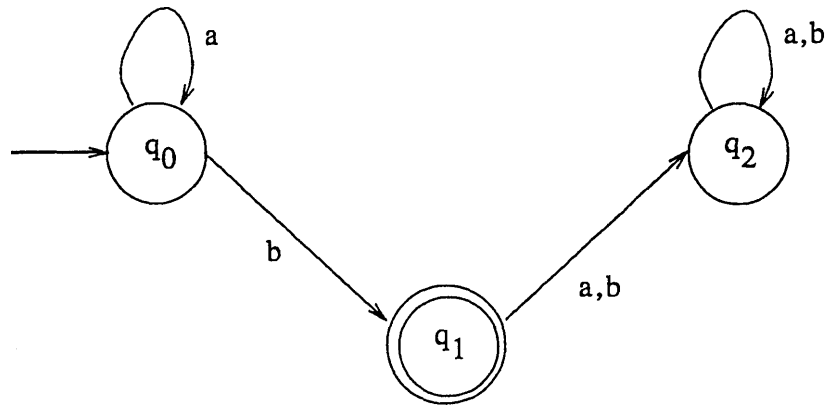


Figure 3.1 The transition diagram of a finite state automaton.

Figure 3.1 shows the transition diagram of this finite state automaton.

The language recognised by the finite state automaton A is $T(A)=\{ x \mid x=a^n b$ with $n=1,2,... \}$, which is the the language generated by the grammar of the example in definition 3.1.

3.3. PROBABILISTIC MACHINES

In this investigation of probabilistic automata, the approach of Rabin [Rabin, 63] is used. There is no deterministic decision as to whether a pattern $\vec{X}$ belongs to the language recognised by a given probabilistic automaton, but only a decision concerning a certain probability of this membership. Languages recognised by probabilistic automata with a cut-point are called weighted regular languages [Salomaa, 69]. Weighted regular languages are generated by weighted regular grammars.

Definition 3.3 - A weighted regular grammar G_w is a 4-tuple $G_w=(V_N,V_T,P_w,S)$ where V_N, V_T and S are as in definition 3.1 above and P_w is

a finite set of weighted productions denoted by $\alpha_i \rightarrow \beta_{ij}$ (p_{ij}) where $\alpha_i \in V_N$, β_{ij} is of the form w or wS_1 with $S_1 \in V_N$ and w is a pattern of terminals, $j=1,\ldots,n_i$, $i=1,\ldots,k$; p_{ij} is a weight associated with the application of this weighted production rule and n_i is the number of productions rules which has the nonterminal S_i as α. The weight associated with a derivation $\alpha_1 \stackrel{*}{\Rightarrow} \alpha_m$ is equal to the product of the weights associated with the sequence of weighted productions used in the derivation.

The weighted regular language generated by G_w is $L(G_w)= \{(x,p(x)) \mid x \in V_T^*, S \stackrel{*}{\Rightarrow} x, j=1,\ldots,k$ and $p(x)=\sum_{j=1}^{k} p_j\}$ where k is the number of distinctively different derivations of x from S and p_j is the weight associated with the j^{th} distinctive derivation of x. The language generated by G_w with a cut-point λ, such that $0 \leq \lambda < 1$ will have the additional restriction of $p(x) > \lambda$ for all patterns generated by G_w.

Below is an example of a weighted regular grammar.

$G_w=(V_N,V_T,P_w,S)$ where $V_N= \{S,S_1,S_2,S_3,S_4\}$, $V_T=\{0,1\}$ and

P_w: (1) $S \rightarrow 1S_1$ $(p=1)$

(2) $S_1 \rightarrow 0S_3$ $(p=0.5)$

(3) $S_1 \rightarrow 0S_4$ $(p=0.5)$

(4) $S_1 \rightarrow 0$ $(p=0.5)$

(5) $S_1 \rightarrow 1S_1$ $(p=0.5)$

(6) $S_1 \rightarrow 1S_2$ $(p=0.5)$

(7) $S_2 \rightarrow 1S_2$ $(p=1)$

(8) $S_2 \rightarrow 0S_3$ $(p=0.5)$

(9) $S_3 \rightarrow 1S_3$ $(p=0.5)$

(10) $S_3 \rightarrow 1S_4$ $(p=0.5)$

(11) $S_3 \rightarrow 1$ $(p=0.5)$

(12) $S_4 \rightarrow 1S_4$ $(p=1)$

(13) $S_4 \rightarrow 1 \quad (p=1)$

The weighted regular language generated by G_w is $L(G_w)=\{(1^m01^n,p(x))|m>0,n\geq 0\}$ and the language generated with a cut-point $\lambda=0.5$ is $L(G_w,\lambda=0.5)=\{(1^m01^n,p(x))|0<m\leq n\}$.

Note that there is no restriction associated with the weights p_{ij}. If it is imposed that $0<p_{ij}\leq 1$ and $\sum_{j=1}^{n_i} p_{ij}=1$ G_w would be a stochastic regular grammar. The set of languages generated by a stochastic regular grammar is the same as that generated by a regular grammar, while the set of languages generated by a regular grammar is a special case of that generated by a weighted regular grammar. That is, there exists a weighted regular grammar which is not a finite state language, for instance, $L(G_w,\lambda=0.5)$ above, is context-free. There is a formal proof that $L(G_w,\lambda=0.5)$ is not regular, in appendix I. One illustration of the relationship between the Chomsky's hierarchy and weighted regular languages is provided in figure 3.2 below, with the following abbreviated notation: WRL is the set of weighted regular languages, RL is the set of regular languages, CFL is the set of context-free languages, CSL is the set of context-sensitive languages and TZL is the set of type zero languages.

This essentially means that: (1) WRL $\supset$ RL; (2) WRL $\cap$ CFL $\neq \varnothing$; (3) WRL $\cap$ CSL $\neq \varnothing$; (4) WRL $\cap$ TZL $\neq \varnothing$;

Definition 3.4 - A probabilistic automaton is a 5-tuple $A_p=(\Sigma,Q,\delta,\pi_0,F)$, where Σ, Q and F are as in definition 3.2 above, δ is a mapping of Σ into the set of nxn (where n is the number of states in Q) probabilistic state transition matrices and π_0 is an n-dimensional row vector and is designated as the initial state distribution. The interpretation of $\delta(a)$, $a \in \Sigma$, can be stated as follows. $\delta(a)=[p_{ij}(a)]$, where $p_{ij}(a)\geq 0$ is the probability of entering state q_j from state q_i under the input a, and $\sum_{j=1}^{n} p_{ij}=1$, for all $i=1,\ldots,n$.

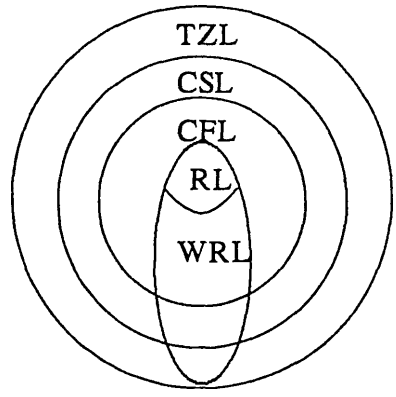


Figure 3.2 Relationship between Chomsky's hierarchy and weighted regular languages.

The weighted regular language accepted by a probabilistic automaton $A_p = (\Sigma, Q, \delta, \pi_0, F)$ is $T(A_p) = \{(x, p(x)) | x \in \Sigma^*, p(x) = \pi_0 \delta(x) \pi_F > 0\}$. The language accepted by A_p with a cut-point λ, such that $0 \leq \lambda < 1$, is $L(A, \lambda) = \{$ $(x \mid x \in \Sigma^*$ and $\pi_0 \delta(x) \pi_F > \lambda\}$, where π_F is an n-dimensional column vector, in which the i^{th} component is equal to 1 if $q_i \in F$ and 0 otherwise.

There is also a one-to-one relationship between the languages generated by regular weighted grammars and the sets accepted by probabilistic automaton.

Below an example of a probabilistic automaton is given.

$A_p = (\Sigma, Q, \delta, \pi_0, F)$ where $\Sigma = \{0,1\}$, $Q = \{q_1, q_2, q_3, q_4, q_5, q_6\}$, $\pi_0 = [1\ 0\ 0\ 0\ 0\ 0]$, $F = \{q_5\}$ and δ is given in the tables below.

	$\delta(0)$					
-	q_1	q_2	q_3	q_4	q_5	q_6
q_1	0	0	0	0	0	1
q_2	0	0	0	.5	.5	0
q_3	0	0	0	.5	0	.5
q_4	0	0	0	0	0	1
q_5	0	0	0	0	0	1
q_6	0	0	0	0	0	1

	δ(1)					
-	q_1	q_2	q_3	q_4	q_5	q_6
q_1	0	1	0	0	0	0
q_2	0	.5	.5	0	0	0
q_3	0	0	1	0	0	0
q_4	0	0	0	.5	.5	0
q_5	0	0	0	0	1	0
q_6	0	0	0	0	0	1

Figure 3.3 shows the transition diagram of this probabilistic automaton.

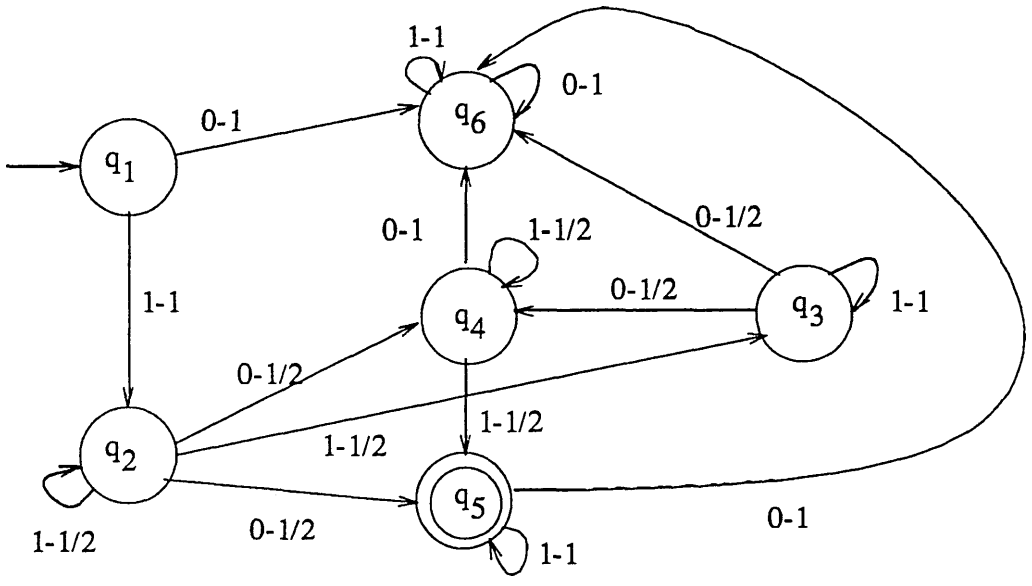


Figure 3.3 The transition diagram of a probabilistic automaton.

The language recognised by this probabilistic automaton is the same language generated by the weighted regular grammar in the example of the definition 3.3, irrespective of whether or not there are cut-points.

3.4. OVERVIEW

In this section, rather than giving a complete chronological review of the history of research on probabilistic automata only the important events and discoveries which are directly related to the concepts in this thesis are outline.

Following the McCulloch-Pitts modelling of the neuron, Kleene [Kleene, 56] investigated the capabilities and limitations of automata constructed from

these idealised neurons. He was interested in what kind of events are capable of being represented in the state of such automata. His principal result was to show that only and all regular languages (definitions 3.1 and 3.2) may be represented by a network composed of McCulloch-Pitts neurons. Thus McCulloch-Pitts neurons are one example of a kind of "universal element" for finite automata. It is of course essential to his arguments that the number of cells and the number of states of each are fixed in advance. An assumption of his mathematical theory, the neurons are reliable and never fail. If the neurons could fail the machine would be non-deterministic.

In order to find a closer approximation to reality von Neumann [von Neumann, 56] made the assumption that with every basic organ is associated a positive number p such that in any operation the organ will fail to function correctly with the probability p. In a complicated network, with long stimulus-response chains, the probability of errors in the basic organs make the responses of the final output unreliable unless some control mechanism prevents the accumulation of these basic errors. Deterministic automata are not suitable for describing the von Neumann model because of the probability associated with the function of each organ. Since the time of the introduction of probabilistic elements by von Neumann, many studies of the consequences of such elements have been published.

De Leeuw et al. [de Leeuw et al., 56] discuss the question 'Is there anything that can be done by a machine with random elements which can not be done by a deterministic machine?'. They defined, the p-machine, a machine composed of random devices, which they analysed in relation to enumerable events. Until that time machines composed of random devices were similar to non-deterministic automata. Later Rabin [Rabin, 63] defined probabilistic automata and showed that these automata can recognise more than finite state

languages. To the concept of random element machines, he added, the idea of having a cut-point λ related to each machine. The class recognised by the machine may change accordingly with the change of the cut-point λ. This development was the inspiration for much further work.

Paz [Paz, 66] made a deep analysis of Rabin's suggestion and showed, among other things, that even with the simplest case (i.e. probabilistic automata with a single symbol in the alphabet and any number of states, or probabilistic automata with any number of symbols and two states) there are events definable by probabilistic automata which are not regular (regular in the sense of the definition 3.1). He also extended a little Rabin's results in stability[1] of probabilistic automata.

The concept of stability plays a fundamental role in the theory of probabilistic automata. The behavioural stability of a probabilistic automaton, A_p, can be defined as follows:

Let A_p have a probabilistic state transition δ and a cut-point λ then A_p is stable if, when slightly perturbed the probabilistic state transition of A_p to give a new automaton A'_p with a probabilistic state transition δ', $T(A_p,\lambda)$ is still equal to $T(A'_p,\lambda)$; that is, the set $T(A_p,\lambda)$ is unchanged by small changes in δ. The physical background to this question related to the circuit theory is obvious. In an unreliable circuit it is not possible to know the probabilistic state transition with absolute accuracy therefore, it is useful to know that even approximate values of δ will lead to a correct description of the circuit behaviour. There are some sufficient conditions for stability and there are cases in which stability is not possible but the general problem is still wide open.

---

[1] Stability in probabilistic automata, in general, means that the set recognised by a given probabilistic automaton does not change under the influence of small perturbations of the transition probabilities of such probabilistic automata.

Salomaa [Salomaa, 69] demonstrated the equivalence between the languages generated by a weighted regular grammar and the languages accepted by probabilistic automata. Fu and Li [Fu-Li, 69] studied weighted regular languages (λ-stochastic as they called, λ is the cut-point) (definitions 3.3 and 3.4 in this chapter) and their relationship with Chomsky's hierarchy. Later, Fu in his book about syntactic pattern recognition [Fu, 82] applied stochastic languages to syntactic pattern recognition. Turakainen demonstrated that for every weighted regular language it is possible to find a probabilistic automaton with a cut-point λ, for any $0<\lambda<1$, which can recognise L [Turakainen, 68]. This means for example, that every weighted regular language is $\frac{1}{2}$-weighted regular. The restriction $\lambda>0$ is essential, because every 0-weighted regular language is regular. Conversely, every regular language is 0-weighted regular because every finite deterministic automaton can be rewritten as a probabilistic automaton, where the probabilistic state transition matrices consist of 0's and 1's only.

The concept of continuous time was applied to probabilistic automata by Knast [Knast, 69b]. This concept is similar to a Markov chain with a continuous parameter. These results can be applied to neural net machines where the state of the machine can change at any time. Knast [Knast, 69a] also introduced the idea of a linear probabilistic sequential machine. He showed that there are linear probabilistic automata which can accept non-regular events.

Page [Page, 69] extended the Rabin and Paz's stability analysis of probabilistic automata. He studied the stability problem for probabilistic automata which are defined from all initial distributions rather than a fixed initial state as Rabin did. The results he obtained are useful when analysing the stability of PLN nets.

Implementations of probabilistic automata are usually based on deterministic devices ([Guiasu, 68], [Paz, 71] and [Fu, 72]). This thesis is concerned with the possibility of implementing probabilistic automata with PLN nets, which will result in a probabilistic implementation of probabilistic automata.

One motivation for studying stochastic automata was the possibility of using them as models of learning and pattern recognition systems. Some work was done by Tsetslin with a deterministic automaton subject to a probabilistic training process [Tsetslin, 61]. Tsetslin was followed by many authors, such as Bush and Mosteller [Bush-Mosteller, 55], Bruce and Fu [Bruce-Fu, 63] and others, who extended and generalised his approach.

There are many other applications of probabilistic automata, as for example: in information theory where the communications channels can be represented as a stochastic sequential machine; in reliability problems, when a deterministic automaton has some unreliable elements, making its external behaviour probabilistic; in control theory with control systems being modelled by stochastic machines with input symbols representing commands; in analysis of programming languages by probabilistic context-free languages, and so on.

3.5. THE APPLICATION OF NEURAL NETWORKS TO FORMAL LANGUAGE RECOGNITION

There has been a great deal of interest in teaching neural networks to recognise grammars and simulate automata. Much work has been done in this field, mostly with MCP networks. Networks composed of MCP neurons are finite state machines but it is not easy to train such networks to recognise regular languages. It is even difficult to make these networks capable of recognising context-free languages; it is only possible with the use of a

memory associated with the network. Such memory can be a stack or an additional weight structure in a multi-layer network. Among the work done with MCP networks in this field, two have been chosen to be described here. The first example demonstrates the state of the art, and the second shows what could be seen as an unprofitable application to neural networks.

Giles et al has defined a single layer recursive network which can learn to simulate a deterministic finite state automaton [Giles et al, 90]. When this neural network state automaton is connected to an external analog stack memory, the combination can be interpreted as a neural network pushdown automaton. The network finite state automaton is given the primitives, push and pop, and is able to read the top of the stack. Through a gradient descent learning rule derived from the common error function, the hybrid network learns to use the stack actions to manipulate the stack memory and to learn very simple context-free grammars. A question which remain open in this work is how well such architectures will scale for more complex context-free grammars which are used in 'real' problems.

Lucas and Damper proposed an algorithm which map, what they call, a non-stochastic strictly hierarchical context-free grammar directly onto a connectionist architecture using a relatively small number of neurons [Lucas-Damper, 89]. What they call a non-stochastic hierarchical context-free grammar is nothing more than a finite regular language. They assume it is possible to infer an adequate non-stochastic strictly hierarchical context-free grammar for the set of patterns they want to recognise. A network is derived with exactly the required number of neurons to match the problem from the inferred grammar; no guesswork is involved. It is not mentioned in their work how well the algorithm is likely to work for more complex grammars. Actually the algorithm proposed in their paper is not able to deal with grammars

which have recursive productions rules.

It will be proved in chapter 5 that logical neural networks have the same computability power as probabilistic automata. However, it is very hard to train these networks to be able to recognise all weighted regular languages. In this thesis an algorithm is given which leads to the design of a logical network that behaves as a probabilistic automaton. Not much work has been done by others on teaching logical networks to recognise grammars. Indeed the only work found in the literature is where SLAM (described in section 2.3 page 31 of this thesis) networks are taught to recognise a small subset of a regular language.

Tollyfield described a training method which enables a very specific network to function as a finite automaton [Tollyfield, 75]. The state structure of this network is, to some extent, arbitrary, since the state labelling is unimportant, but the transitions must bear some relation to the type of productions permitted. The language which can be accepted by such a network is very restricted. Experimentally he showed that the chosen network appears to accept all languages generated by regular grammars subject to two restrictions: firstly, that the number of non-terminal symbols in the grammar should not exceed three. Secondly, no more than two recursive productions should be defined, one of which should specify the starting symbol S. Also the languages can have only two terminal symbols. These restrictions on the regular languages which can be learned by this method are severe. Although there was scope for extending both the network and the training algorithm to encompass languages generated by more complex grammars no further work has appears to have been done.

3.6. SUMMARY OF THE CHAPTER

This chapter presented a brief survey on the work done in probabilistic automata. Definitions of regular grammars and finite state automata were given in the second part of this chapter. Weighted regular grammars and probabilistic automata were also defined. The relationship between weighted regular languages and languages in Chomsky's hierarchy was illustrated. Examples of regular grammars, finite state automata, weighted regular grammar and probabilistic automata were given. The use of automata theory in neural networks was discussed. Some neural networks designed for formal language recognition were described.

CHAPTER 4

A FEEDBACK RAM-NETWORK FOR TEMPORAL PATTERN RECOGNITION

4.1. INTRODUCTION

In this chapter a system is described which is able to deal with temporal patterns. As anticipated in earlier chapters, the model employed is a feedback neural network based on RAM neurons. It is shown that the response of a RAM network with feedback carries information about the order of appearance of its input patterns. This makes these networks suitable for temporal pattern recognition. Three different ways of doing the recognition are introduced, a probabilistic classifier, a buffer classifier and a finite state classifier. The use of different classifiers gives a better insight into the systems and shows the importance of the RAM network itself. Some experimental results are given with each one of the classifiers used. A comparison of the results obtained with the different classifiers is made. The influence of a stability property[1] in the generalisation of a network and consequently in the ability of the network to solve a specific task is discussed. Some ways of controlling the stability of the network are presented and this is supported by experimentation.

4.2. DESCRIPTION OF THE NETWORK

The type of network used in this work consists of a layer of identical RAM type digital neurons, where each has k-address terminals, i of them con-

[1] When, in this chapter, reference is made to the stability property or stability in a network it is the stability of the network with respect to input sequences which is considered. That is, stability in the tendency for the response of a given network not to change under the influence of small errors in the input sequences. The effect of the stability property in logical neural networks is the increase of the generalisation of the network.

nected to an external matrix of binary elements and f of them connected to the output terminal of other neurons through clocked delay units ($k=i+f$). The structure of a Sequence Digital Neural Network (SDNN) is represented in figure 4.1 where binary vectors $X(t)$, $R(t)$, $R(t-1)$ and $D(t)$ represent respectively at time t, the state of the input matrix, the response of the neurons, the delayed response of the neurons and the 'desired' response of the neurons. Since the input connections are fixed and the digital neurons (RAM) used are deterministic devices, the response $R(t)$ is only function of $X(t)$ and $R(t-1)$:

$$R(t) = h[X(t),R(t-1)]$$

Unless otherwise stated, in the SDNN used in this work, the input vector $X(t)$ has the same dimension (n-bits) as $R(t)$, $R(t-1)$ and $D(t)$. The input and feedback connections are randomly generated.

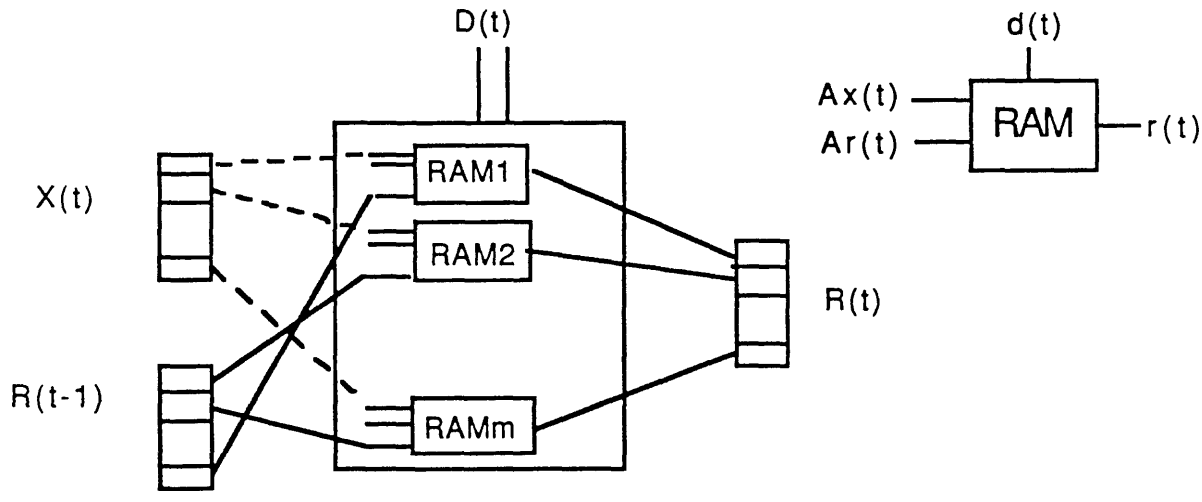


Figure 4.1 Sequential Digital Neural Network.

The notation SDNN(m,i,f) is used for a Sequence Digital Neural Network with m neurons; each neuron has i terminals connected to the external input and f connected to the feedback input. The contents of RAM elements are randomly set before training. The initial state of the network is randomly

generated and the network is always put in the same initial state before feeding a pattern to it.

As illustrated in figure 4.1, the input stimulus of each digital neuron at time t, $A(t)=\{A_x(t),A_r(t)\}$ has two components: $A_x(t)$, the input component with i-bits which depends solely on $X(t)$ and $A_r(t)$, the feedback component with f-bits, depending only on the state of the network, $R(t-1)$. Therefore, the response of a given neuron, $r(t)$ is only a function of its input and feedback components: $r(t) = g[A_x(t),A_r(t)]$

TRAINING THE NETWORK

In general, a training process, which consists of subjecting a SDNN to change in the functions performed by its neurons, involves the choice of the training vector $D(t)$ as a function of both the input ($\vec{X}$) and output ($\vec{R}$) sequences, $D(t)=h[\vec{X},\vec{R}]$.

In the training mechanism used in this work the SDNN is trained to anticipate its inputs, i.e. $D(t)=X(t+\alpha)$, such that $R(t)=X(t+\alpha)$ might be obtained during the test phase. For example, with $\alpha=1$ one has $D(t)=X(t+1)$, the network tries to predict the next input symbol.

The input matrix is represented by a subset T (the training set) of the language L that needs to be recognised (in figure 4.2). A subset of language L is a set of sequences of any length which belong to the language L. A subset can have any number of sequences belonging to the language L.

During the training phase the network is fed with $\vec{X} \in T$ with one (or more) RAM(s) in the write mode, and the contents of the memory position is changed $M_i[A_x(t),A_r(t)]=d_i(t)$ for all RAMs in the write mode.

The choice of the training strategy depends on the desired application. Unfortunately, there is not a developed theory in which, for a certain kind of

problem, a good training strategy can be determined. Here the training strategy $D(t)=X(t+\alpha)$ was chosen with the objective of the recognition of input sequences through the states of the network. The goal with this training strategy was to make the network stable with respect to small changes in input sequences. In section 4.5 the effect of the training strategy described here on the stability of the network will be explained. Another point to stress is the fact that the percentage of RAMs that are made to learn at each moment is variable (i.e. the training rate).

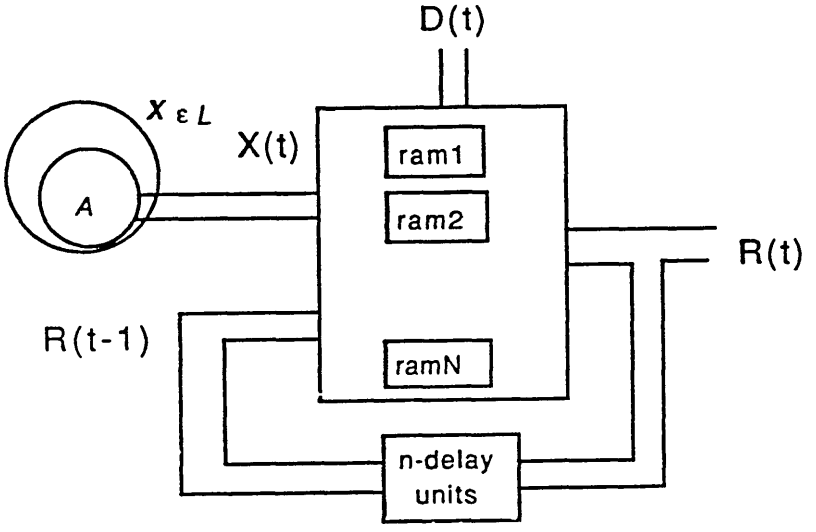


Figure 4.2 The training phase.

The need for controlling the training rate was first realised by Reeves [Reeves, 74] in his attempts to classify image tracking sequences. A low training rate is adopted in the experiments of this thesis, where, mostly, one neuron is allowed to change at each time. A training mechanism that changes the responses of the neurons in a gradual manner is necessary because contradictions exists among the input sequences. With a low training rate the network takes longer to be trained. This process can be speeded up provided one can find an optimal set of training sequences, i.e., a set that is compact and whose

elements are properly representative of their classes (with less contradictions as possible). However, finding such an optimal set is not a straight forward task. Another interesting mechanism that has been incorporated into some training processes, is the AGEING mechanism. AGEING was introduced by Fairhurst [Fairhurst, 73] and consists of allowing a high initial training rate which is reduced as the training phase goes on, eliminating the dominance of the last pattern seen by the network. The problems with AGEING are that the network may not stay long enough under the low training rate, which is more effective than high training rate, and it introduces complications in the training phase such as deciding for how long the training rate should be high and how often and how much the training rate should be decreased. It is much simpler to adopt a constant low training rate for the whole duration of the training phase.

In order to justify the use of a single layer network with feedback in sequence discrimination one needs to remember that the response of SDNN carries information about the order of appearance of its input patterns, that is each output of the network depends not only on the present input but also on the previous ones. This is a well known result of finite state machines [Aleksander-Hanna, 76]. But feedback networks are not the only way to deal with this problem. It is possible to use a feed-forward RAM-network where the chain mechanism created by supplying the information available by the feedback loop is given directly from the external input of the network. For instance, once the training strategy here is $D(t)=X(t+1)$, $R(t)\simeq X(t+1)$, then $X(t+1)$ could be supplied to the network as an input symbol simultaneous with $X(t)$. This is not to say that feedback networks behave like a feed-forward network. The temporal properties of a feedback network are simply not needed to create the chain mechanism. One advantage of feed-forward RAM-

networks is that they are less sensitive to input errors. For example, with a feed-forward network where the external inputs are $X(t)$ and $X(t+1)$, the effect of a single error cannot last longer than 2 units of time. Whereas the same cannot be said about a feed-forward network like that in figure 4.1. Due to the temporal action of the feedback loops, the effects of a single error may persist for much longer than 2 units of time. However, feed-forward networks, because of their structures, are not inherently able to store sequential information from their input patterns.

4.3. NATURE OF THE EXPERIMENTS

This section describes the nature of the temporal patterns whose recognition is investigated in the next sections. Two different sets of temporal patterns are used. The first is generated by the tracking of different geometric forms, such as triangles, squares and circles. The second one is generated by regular grammars. In both cases the temporal patterns consist of segmented or discrete temporal patterns, i.e., a whole consists of well defined segments or components. In contrast, spoken and hand-written words are examples of temporal patterns without segmentation, i.e., the continuous case. The fact that in our examples the patterns are already segmented makes their recognition far easier than that of spoken words. In order to appreciate the importance of segmentation one must remember that the most fundamental mechanism in pattern recognition consists of breaking down complex patterns into sub-patterns that can be easily recognised, called primitive patterns or features. What makes speech recognition such a difficult problem is that the segmentation of continuous speech into phonemes is an extremely complex task. An inherent characteristic of segmented temporal patterns is that they can be handled by synchronous systems. In other words, a segmented temporal pattern

can always be represented by a sequence $X(1),X(2),...,X(t)$ of discrete events or patterns belonging to a finite set of patterns. Since the work reported here involves the use of digital systems, the temporal patterns that are dealt with throughout the whole thesis are, in fact, sequences of binary vectors.

GEOMETRIC FORMS

The data set used in these experiments was generated and first used by Fernandes [Fernandes, 77]. The set consists of tracking movements, of the type used by Freeman [Freeman, 61], for geometric forms drawn on the screen of a graphic display. The tracking consists of moving a cursor from a point of a grid in the direction of its eight neighbours. This can be seen in figure 4.3 below. In order to feed the network with a sequence of moves, in every instant of time t, every move was coded as a binary vector. Experiments were done with the tracking sequences generated from circles, triangles and squares.

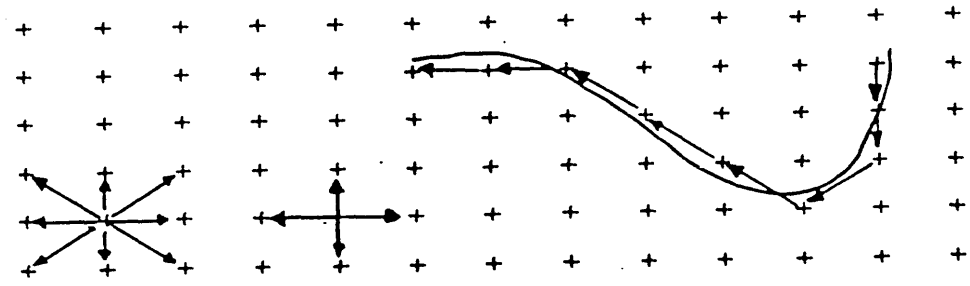


Figure 4.3 Tracking Movements.

REGULAR LANGUAGES

The sequences in this set of experiments were generated by different regular grammars. The symbols of the alphabet were coded as binary vectors.

Separate training and testing sets of sequences generated by the grammar were used. Also a testing set of sequences generated by the complement grammar was used.

The final configuration of the network is a finite state automaton. The aim of this group of experiments was to analyse the behaviour of the network when recognising different regular languages.

4.4. PROBABILISTIC CLASSIFIER

This section describes a classifier based on probabilities of occurrence of patterns in each of the two classes, L_1 and L_2, that need to be distinguished. If one wants to distinguish more than two classes one can combine many of the systems suggested in this chapter. The a posteriori probability, as defined below, is used in this classifier.

The information carried by an input symbol X_i regarding the class membership of an input sequence $\vec{X}$ is directly related to the frequency with which X_i occurs in sequences of both languages L_1 and L_2. This rather quantitative statement is a well known fact from information theory [Shannon-Weaver, 49]. A classifier based, then, on the probability $p(\vec{X} \in L_k/\vec{R})$ that the input sequence $\vec{X}$ belongs to the language L_k, given that the response $\vec{R}$ occurred when $\vec{X}$ was fed into the network makes use of the information related with the occurrence of the symbols in each class. If $p(\vec{X} \in L_1/\vec{R}) = p(\vec{X} \in L_2/\vec{R})$, the response $\vec{R}$ carries no information whatsoever regarding the class membership of $\vec{X}$. $p(\vec{X} \in L_k/\vec{R})$ is called the *a posteriori* probability. In general, the greater the difference $|p(\vec{X} \in L_1/\vec{R}) - p(\vec{X} \in L_2/\vec{R})|$ the more information $\vec{R}$ is carrying about the class membership of $\vec{X}$.

Although the ideal would be $p(\vec{X} \in L_k/\vec{R}) = 1$ for all symbols in the

sequence $\vec{X} \in L_k$, this does not generally happen. What often happens is that $p(\vec{X} \in L_k/\vec{R})$ changes randomly near 1 for sequences $\vec{X}$ in L_k and near 0 for sequences $\vec{X}$ not in L_k. Hence, a measure is required which considers $p(\vec{X} \in L_k/\vec{R})$ for all input symbols in the same sequence $\vec{X}$. The measure that was used was the continuous average

$$S_k(t) = a^{-1} \sum_{t=1}^{a} p(\vec{X} \in L_k/R(t)),\ k = 1,2 \text{ and } 0 \leq S_k \leq 1.$$

Using this measure, for all sequences belonging to the language L_k the sum $S_k(t)$ tends to 1, while for all sequences not belonging to the language L_k the sum $S_k(t)$ tends to 0.

Note that the network used in combination with the classifier has output and feedback state very big (in many experiments of this thesis they have 32-bits). This means that a huge number of different states are possible (2^{32} in the case of a 32-bits state). It is not likely that the network will follow the same sequence of states when a different sequences of input are fed to the network even if the different sequences have the same symbols with the same frequency.

For instance if two different sequences $\vec{X}$ and $\vec{Y}$ are fed to the network. $\vec{X} = ABC$ and $\vec{Y} = ACB$, where A, B and C are the input symbols of the different languages and are coded in 32-bits. Suppose that the network before the feeding of both $\vec{X}$ and $\vec{Y}$ are in an initial state R_0. Suppose now that the output states for $\vec{X}$ are R_1, R_2 and R_3 and for $\vec{Y}$ is R_1, $\bar{R}_2$ and $\bar{R}_3$. So it is not likely that, with such a big number of possible states (2^{32}), R_2 will be equal to $\bar{R}_2$ and R_3 will be equal to $\bar{R}_3$. If, on the other hand, the sequences $\vec{X}$ and $\vec{Y}$ were not fed to the network but the classifier was used with the $\vec{X}$ and $\vec{Y}$, no discrimination would be possible with this classifier. $\vec{X}$ and $\vec{Y}$ have exactly the same input symbols with the same frequency and this classifier is only based on the frequency with which the symbols occur in each class.

It is assumed that in coding the symbols the biggest Hamming distance between them as possible exists because it is always possible to choose the code in a way that this is true. Experiments were carried out using only the classifier, without the network. The results showed, of course, that no distinction at all between the classes which have the same set of input symbols was possible and where the frequency of occurrence of the symbols in different classes is similar.

The approach proposed here was a discriminating one where the sum S_k carried only the amount of sequential information (or structural description) necessary to discriminate sequences $\tilde{X} \in L_1$ from sequences $\tilde{X} \in L_2$. Since the transformation $f : \tilde{X} \rightarrow S$ was based on a "sequential sample" of the input sequence $\tilde{X}$, the discrimination technique examined was not very sensitive to distortion over input sequence variations. The presence of errors, like deletion and insertion of symbols did not impair the discrimination process [Fernandes, 77].

The system proposed here is composed of three phase:

Phase One: Training the network. In this phase the network is trained with sequences $\tilde{X}$ belonging to the language L to be recognised as explained in section 4.2.

Phase Two: Calculating the probabilities. In this phase the a posteriori probabilities for sequences $\tilde{X}$ belonging to the language L to be recognised $p(\tilde{X} \in L_k / \tilde{R})$ are calculated and stored to be used in the next phase.

Phase Three: Testing the system. The input sequences are fed to the network and based on the probabilities $p(\tilde{X} \in L_k / \tilde{R})$ the sums $S_k(t)$ are calculated.

This system could have an extra phase where a threshold would be calculated based on the sums for different languages to be recognised. In such case the recognition of the sequences would be done based on the value of the

threshold. The experimental results shown here are related to phase three.

Below the results of the experiments to distinguish different geometric forms given their tracking sequences are given. The network was composed of 32-RAMs with four inputs each, two from the input patterns and two from the feedback state. The input patterns $\vec{X}$ correspond to the sequence of tracking movements. The first feedback state was randomly generated.

In the figure 4.4 the language L_1 was composed of triangles of different sizes and L_2 was composed of squares and circles as examples of non-triangles. In the figure 4.5 the the language L_1 was composed of squares and L_2 composed of triangles and circles as examples of non-squares.

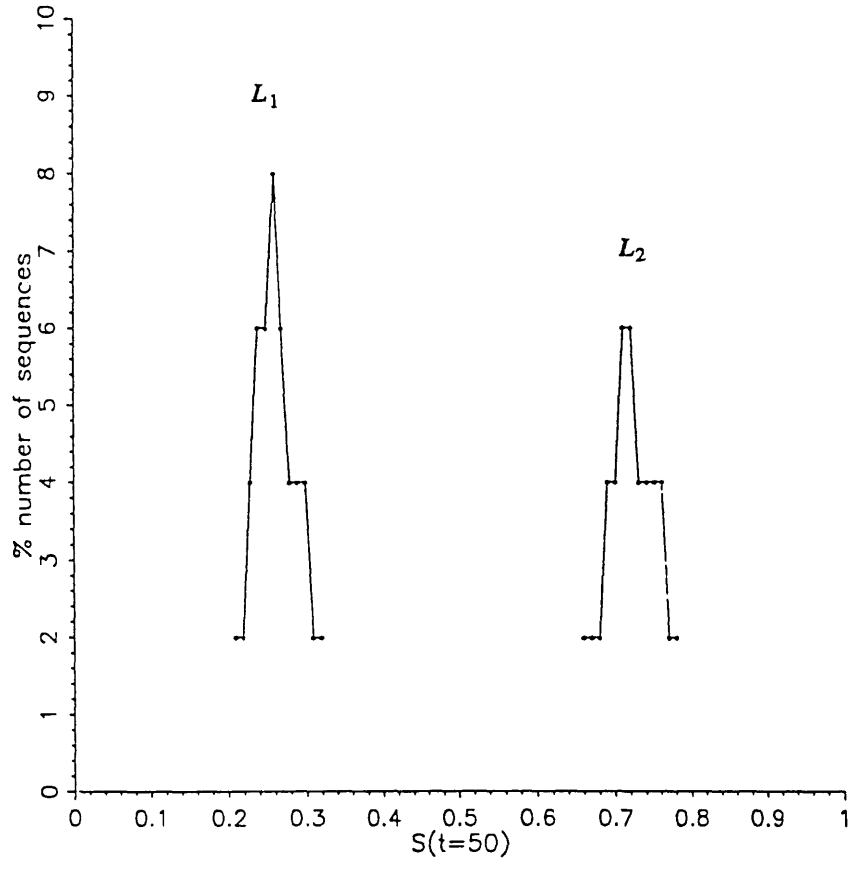


Figure 4.4 Probabilistic Classifier with L_1=triangles and L_2=squares and circles.

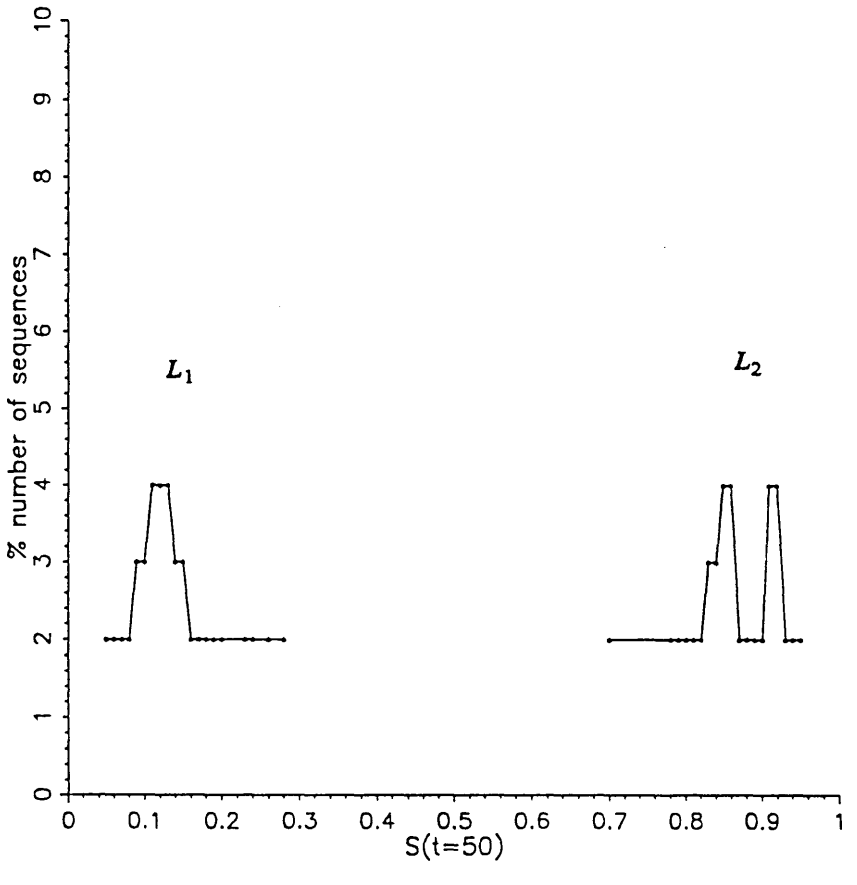


Figure 4.5 Probabilistic Classifier with L_1=squares and L_2=triangles and circles.

This classifier was effective at distinguishing different geometric forms. The leftmost curves in the figures are for sequences $\tilde{X}$ in L_2 while the right-most curves are for sequences $\tilde{X}$ in L_1. Note that if ones chooses a threshold between 0.4 and 0.6 in figure 4.4 and between 0.3 and 0.65 in figure 4.5 one is able to distinguish the two classes.

Figure 4.6 shows the results of an experiment with regular languages. In the experiment L_1= $\{\tilde{X}|\tilde{X}=a^ib^jc^kd^l, i,j,k,l\geq 0\}$ and $L_2=U-L_1$, where U is the universal set.

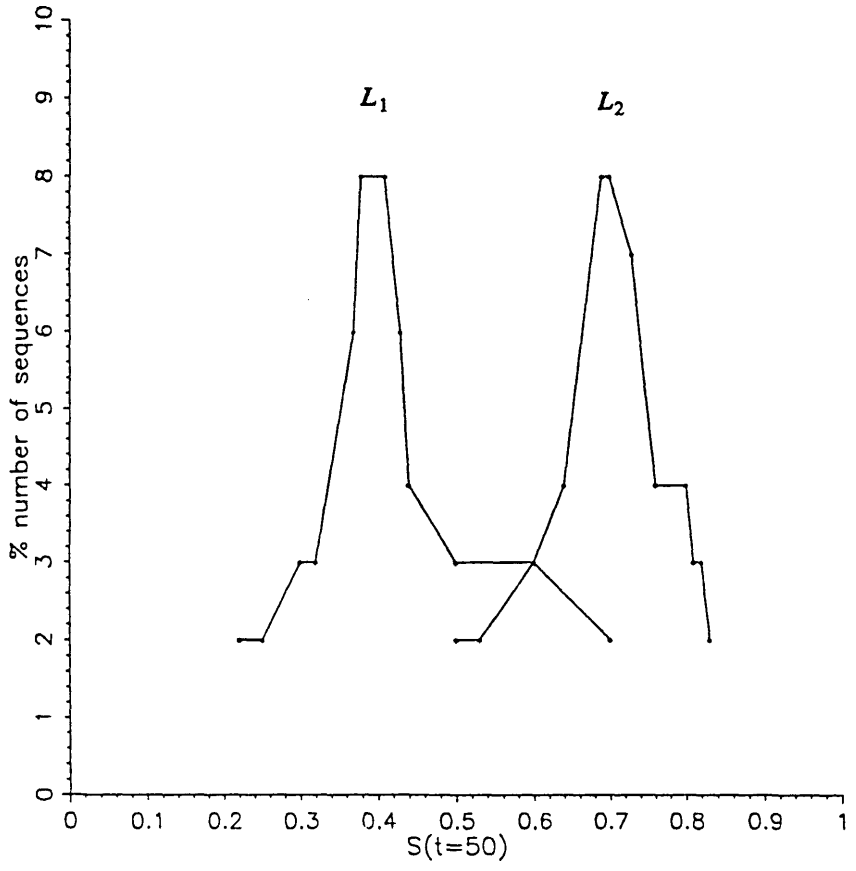


Figure 4.6 Probabilistic Classifier with $L_1 = \{\vec{X} | \vec{X} = a^i b^j c^k d^l, i,j,k,l \geq 0\}$ and $L_2 = U - L_1$.

Although this classifier was effective for distinguishing different geometric forms, it was not efficient with regular languages. Any value chosen for the threshold leads to some misclassification. One of the reasons for this is that in such regular languages the position of the symbols in the sequences are crucial for the sequence to be classified as belonging to a language. This means that a great deal of sequential information should be stored by the network and knowledge of more previous inputs needs to be processed all the time by the system. Consequently the design of a buffer classifier will be described, which was able to deal with more sequential information. Also the influence of the stability property on the generalisation of the network was studied and this is followed by a consideration of the performance of the network to solve a specific task.

4.5. STABILITY PROPERTY

For most practical applications, one has to deal with error and distortions over which there is no control. This is the case for transmission over noisy channels and also for the recognition of speech and other pattern recognition tasks. Using RAM- networks it is easily seen how the stability (the effect of input errors) is controlled by the choice of a well-defined class of functions performed by the neurons.

The stability property of the network is responsible for the increase of generalisation. Thus it has direct influence on pattern discrimination and identification. The relationship between the stability and the generalisation of a feedback RAM-network was studied with the aim of improving the results with temporal pattern recognition tasks. Also the influence of different stability control methods in relation to sequence recognition was examined.

The stability of RAM-nets mainly depends on two parameters: a) neuron memory contents and b) feedback connection.

a) Neuron memory contents. The stability can be controlled by the distribution of 0's and 1's in the neuron memory. The difference between the number of 0's and 1's in the neuron memory is denoted by DL. The greater the difference DL between the number of 0's and 1's in the neuron memory the less will be the probability of changes in the neuron output, in consequence the network will be more stable. There are two different ways of controlling the memory contents. 1) direct control and 2) adaptive control. These were first used by Fernandes in [Fernandes, 85] and then by Ludermir [Ludermir, 86].

1) Direct control. The functions of each neuron is selected at random in such a way that a particular difference DL could be the same for all neurons.

2) Adaptive control. The distribution of 0's and 1's is made through training with any training strategy.

b) Feedback connection. These influence the recovery from an input error in time in a network. If the network has a low number of feedback connections the error propagation through time is reduced and the recovery phase will be short. Thus the network is more stable. The greater the number of feedback connections the larger the influence of error on the network.

The measures used to analyse the results of experiments on stability was: the size of state sets in each class of sequences fed into the network; the percentage of error recovery of the input sequence; and the amount of sequence distinction. The state sets of each class should not be large (in order to save process time and memory) nor that the common states should be many (in order to have good discrimination). Take two input sequences $\tilde{X}=X(1)X(2)...X(t)$ and $\tilde{X}'=X'(1)X'(2)...X'(t)$, where $\tilde{X}'$ is a distorted version of $\tilde{X}$ such that they have few different input symbols. In other words $X(t)\neq X'(t)$ for some instant of time. The sequence $\tilde{X}$ is used at training phase and $\tilde{X}'$ is used at test phase. The percentage of recoveries from errors in an input sequence influences the size of the state set in each language. That is, once the Hamming distance between the responses of the sequence $\tilde{X} \in L$, fed into the network during training, and the test sequence $\tilde{X}' \in L$ is zero $(h(t)=h[r(t),r'(t)]=0)$ all the responses (states) will be the same. Networks which are stable will generate a smaller quantity of different states than networks which are not stable.

Experiments were done with different geometric forms and different networks for which the feedback connections were changed ([Ludermir, 89b] and [Ludermir, 90a]). The networks used were SDNN(32,2,2) (a Sequence Digital Neural Network with 32 nodes; each node has 2 terminal connected to the external input and 2 connected to the feedback input), SDNN(32,2,3), SDNN(32,2,4), SDNN(32,2,5) and SDNN(32,2,6). The distribution of 0's

and 1's in the neuron memory were controlled by the direct and adaptive control methods described earlier. The goal with the experiments here is to show the relationship between the stability of the network and its capacity to discriminate patterns. The feedback connection of the network are increased in the experiments because as high it is the feedback connection as less stable the network is. The network is less stable with high feedback connections because the changes in an input pattern will take longer to be forgotten by the network, generating different responses until the network has completely forgotten the difference between the input patterns. Another way to make networks stable is by controlling the differences between 0's and 1's in the memory of the nodes. The extreme case is when all the memory has only 0's or only 1's. In such cases the network will give the same answer for all input patterns (complete stability).

ERROR RECOVERY OF INPUT SEQUENCES

When the networks are more stable there will be a better recovery from the input errors because the network will not give give different responses to small differences in the input patterns. It is expect, then, that the network with small number of feedback connections will have better recovery from the input errors (networks with small number of feedback connections are more stable than networks with big number of feedback connections). It is important to note that networks which have a large difference between the number of 0's and 1's in the memory of their RAMs, resulting from direct control of stability, do not always display a high recovery rate from input errors. This is because the increase in the differences between 0's and 1's is made randomly. For similar patterns the direct control can give completely different response because the network is not shown the patterns and their similarities. This does not happen with the adaptive control where the changes in the memory con-

tents are done by training and the network is exposed to the common characteristics of the patterns in the same class.

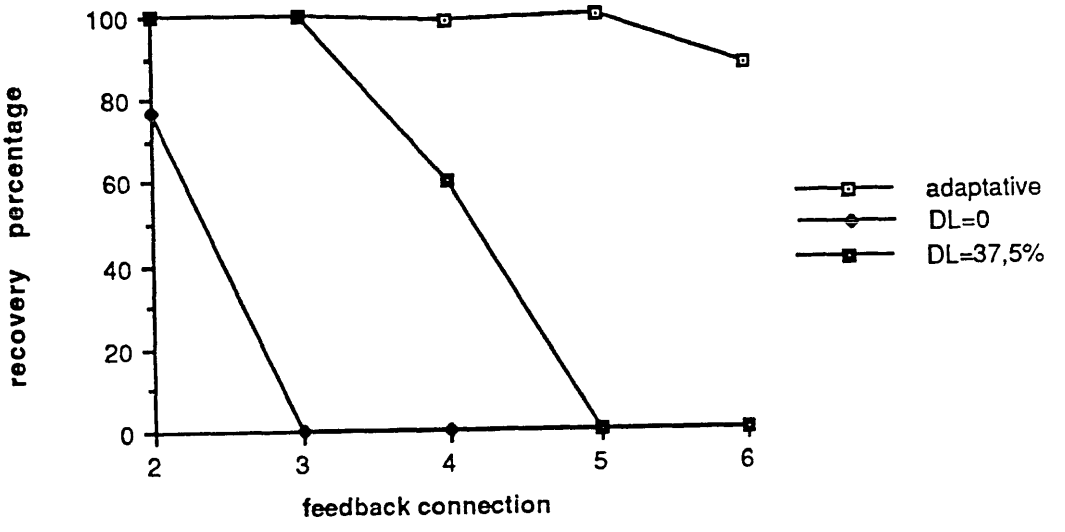


Figure 4.7 Recovery Percentage

Figure 4.7 shows the percentage of error recovery of input sequences for experiments with five different networks. One can see that when the number of feedback connections increases the percentage of error recovery of input sequences decreases. With SDNN(32,2,2) there is no difference between a direct control of stability with high value of DL and adaptive control because the number of feedback connections is small and as a consequence the network is very stable. The network SDNN(32,2,3) with direct control and with $DL = 0$ (that is: half of the memory contents are 0's and half are 1's) was not able to recover from input error at all because the network was made unstable from the very small difference between the number of 0's and 1's and also from the increase of the number of feedback connections. Networks with greater number of feedback connections were less stable and only the adaptive control method was successful in making these networks able to recover from input errors because the direct control also makes the network not stable.

SIZE OF STATE SETS

When the networks are more stable the size of state sets will be smaller than when the network are not very stable because when the network is unstable even similar patterns will generate different states. It is expected, then, that networks with small number of feedback connections will generate small size of state sets. The maximum number of states by class is the smaller value between: 1) 2^n, where n is the number of bits in the output response (state) or 2) the number of sequences fed into network multiplied by the number of symbols in each sequence. The size of state sets has an effect upon sequence discrimination. In experiments in which the size of the state sets is very large there is no discrimination between sequences belonging to different classes because most states belonging to all different classes.

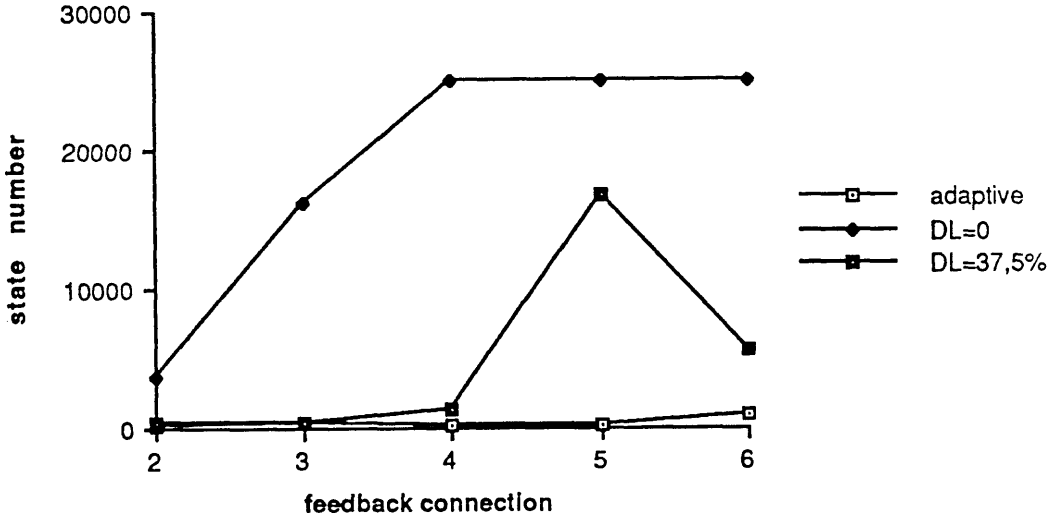


Figure 4.8 State Sets Size

Figure 4.8 shows the number of states in each class of sequences for experiments with the five different networks. The maximum number of states by class is the number of sequences fed into the network, multiplied by the number of symbols in each sequence: 500×50=25.000. SDNN(32,2,2) using

adaptive control produced a smaller state set, that is, less computation was necessary to discriminate sequences in different classes, than when using direct control. With SDNN(32,2,3) the size of the state sets was closer with adaptive control and direct control with high value of DL. Networks with greater number of feedback connections produced very large state sets with direct control, whilst with adaptive control they were still of a reasonable size because both large number of feedback connections and direct control make the network unstable.

SEQUENCE DISCRIMINATION

With more stable networks the discrimination of sequences in different classes is better because the number of states in each class is not big and the recovery from input errors is good. With an unstable network the discrimination of a patterns is difficult because small variations between the prototypes (patterns in the training set) and the test pattern result in a completely different response sequence $\tilde{R}$. An unstable network generates more unknown rejections whilst a very stable one generates more rejections due to low confidence (the network is unable to notice the differences between patterns) and errors from too much generalisation. The generalisation of stable networks are greater than the generalisation of unstable ones. This implies that with more stable networks there are more patterns not in the training set being recognised. It is necessary to limit the size of the generalisation set because large generalisation sets generate more recognition error.

Figure 4.9 shows the discrimination capability of the network for experiments with five different networks using the probabilistic classifier and the geometric forms as data. The discrimination capability of the network is illustrated by the difference between the values of $S_1(t)$ from sequences in the class to be recognised and the values of $S_1(t)$ from sequences which do not

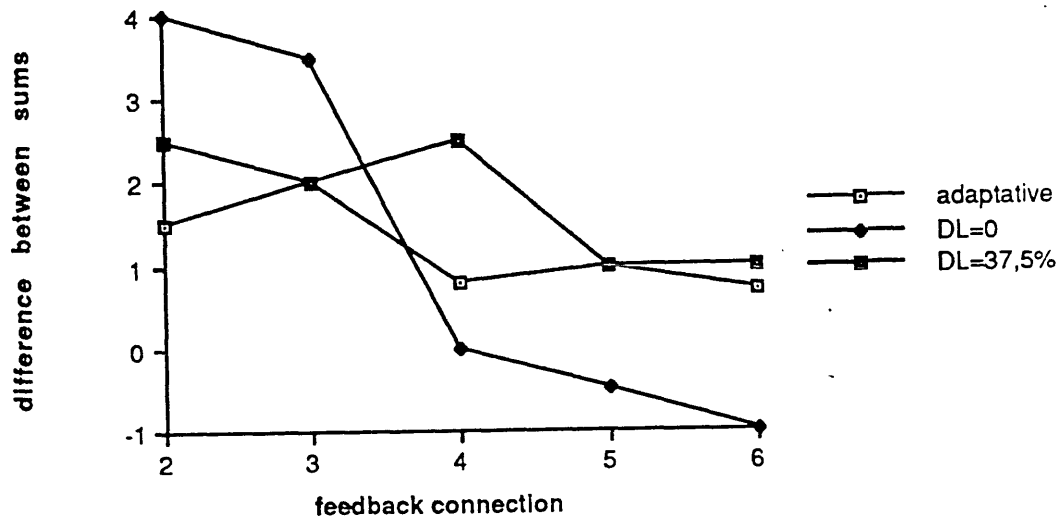


Figure 4.9 Sequence Discrimination

belong to the class to be recognised. When the values of the difference between $S_1(t)$ for $\bar{X}$ belonging to L and $S_1(t)$ for $\bar{X}$ not belonging to L are shown to be negative in figure 4.9, it means that there was an intersection between the values of the sums as in the case of the experiment showed in figure 4.6. There, the difference is -2, which means that there is an intersection between the values of the sums and that it would be a misclassification if the sums are in a certain range of .2. If the sums are between .5 and .7 it is not possible to decide to which class the sequence belongs. With SDNN(32,2,2) and SDNN(32,2,3) it was possible to distinguish the sequences with both ways of controlling the stability of the network because the networks were stable in all cases and the state sets were not very large.

It should be noted that with SDNN(32,2,2) there was not too much difference in the rate of input error recovery between the two methods of stability control, and that the size of the state sets with adaptive control was smaller than with direct control; meaning that the adaptive control is the more efficient in sequence discrimination. However with SDNN(32,2,3) the size of

the state sets was similar with adaptive control and direct control with high value of DL; the input error recovery was similar and the discrimination power between the two ways of controlling the stability of the networks was also the same. Networks with more number of feedback connections had values of $S_1(t)$ which were very close to each other mainly in the cases where the stability control was direct with $DL=0$. With a high number of feedback connections the network was not very good at discriminating sequences. Increasing the number of feedback connections causes the network to remember an input error for a longer time resulting in a less stable structure. The percentage of input error recovery deteriorates. State sets become larger and in some cases this will cause difficulties in performing temporal pattern recognition.

Below is given a short description of the effect of the training strategy described at the section 4.2 on the stability of the network. Given two different instants of time t_i and t_j, a given neuron may be subjected during a teaching process, where the stored responses are changed according to $M[A(t)]=d(t)$, to one of the following conditions:

$$d(t_i)=d(t_j) \text{ with } A(t_i)=A(t_j) \quad \text{eq.(a)}$$

$$d(t_i)=d(t_j) \text{ with } A(t_i)\neq A(t_j) \quad \text{eq.(b)}$$

$$d(t_i)\neq d(t_j) \text{ with } A(t_i)=A(t_j) \quad \text{eq.(c)}$$

$$d(t_i)\neq d(t_j) \text{ with } A(t_i)\neq A(t_j) \quad \text{eq.(d)}$$

Equation (c) has clearly a contradictory effect, for the neuron is asked to store different responses for the same stimuli, whereas equation (b) has the opposite effect, since for two different stimuli the neuron associate the same response. If equation (b) occurs often in the training phase the number of dif-

ferent states will decrease because for different input patterns, the same state is being associated. On the other hand if the equation (c) occurs often the number of states will increase because the same input is associated with different states. Considering the training conditions above, it can be said that a SDNN can only become more stable, due to training, if the condition stated by equation (b) occurs more frequently than that expressed by equation (c). As explained before stable networks generate small number of states.

4.6. BUFFER CLASSIFIER

This classifier was designed to make the discrimination system capable of processing more sequential information. With the probabilistic classifier described in section 4.4 it was not possible to distinguish sequences from the second set of experiments (Regular Language Data Set).

The RAM-feedback network used in this thesis was blind to small differences in patterns. This can be explained through the stability property. Take two sequences $\vec{X}=X(1)X(2)...X(t)$ and $\vec{X}'=X'(1)X'(2)...X'(t)$ where $\vec{X}'$ is a distorted version of $\vec{X}$ such that: $X(t)\neq X'(t)$ for $t=1$ and $X(t)=X'(t)$ for $t>1$, for instance. $\vec{R}=R(1)R(2)...R(t)$ and $\vec{R}'=R'(1)R'(2)...R'(t)$ denote the corresponding output sequence with respect to $\vec{X}$ and $\vec{X}'$ respectively. Since the Hamming distance $h[X(1),X'(1)]>0$ then naturally $h[R(1),R'(1)]>0$. But $h[R(t),R'(t)]$ tends to decrease as t increases due to the fact that $h[X(t),X'(t)]=0$ for $t>1$. The SDNN tends to enter the output sequence $\vec{R}$ when t increases. In other words a SDNN tends to forget the differences between the input sequences $\vec{X}$ and $\vec{X}'$.

The feedback states of the RAM-network used in the probabilistic classifier do not contain enough information to recognise small differences between patterns. In order to save more information a buffer was introduced

into the probabilistic classifier. The buffer stored the time of occurrence of the states.

One state can occur with the same probability to patterns in different classes which makes this state to carry no information about the class of the patterns. But if this state occurs in different instant of time in the different classes and the time is stored, then this state carries information about the class of the patterns. This classifier is composed of the same three phases of the probabilistic classifier. It is in phase two that the time of occurrence of the states are stored. For each language there will be, k sets of states, where k is the longest size of sequences in the languages. Here k is equal to 50. Also at any instant of time there will be two set of states, one to each language, to be considered.

When feeding different sequences $\tilde{X}$ in the same language to the network, it is not likely that the states (or the responses $R(t)$ of the network) generated by all the sequences of a language appear in the same order. From this fact, a window in which the states can happen is defined. That is, the states are allowed to occur in an interval of time but not at any time. This is the same as to say that the responses (states) of the network are allowed to occur in some different order from the responses generated by the training sequences but not in any order. If the responses can occur in an interval of time it means that such response can occur in any time in that interval which implies that the responses generated by an input sequence can occur in a different order in a specific interval. How different it can be the order of the responses generated by an input sequence is controlled by the size of the window. If the size of the window is the length of the longest sequence in the language, this classifier becomes the probabilistic classifier. If the size of the window is equal to 1, the states need to occur in exactly the same order as that

in which they occurred during the training phase. If the state occurs during training more than once in the window, the probability chosen for the state is one instance of time closer to the instant of time at which the state occurred during the test phase.

In the same way as in the previous classifier the ideal would be $p(\vec{X} \in L_k/\vec{R})=1$ for all symbols in the sequence $\vec{X} \in L_k$, but this does not generally happen. Again, the continuous average of the probabilities is used. The sum is defined as:

$$S_k(t)=a^{-1}\sum_{t=1}^{a} p(\vec{X} \in L_k/R(t'))$$

where $t'=t\pm t''$, $t''=0,1,...,ws$; $k=1,2$ and $0\leq S_k\leq 1$ with ws being the window size.

This classifier was submitted to the same experiments as with the probabilistic classifier. Better results were achieved with the buffer classifier than with probabilistic classifier because extra memory were added to the buffer classifier and the time of occurrence of the responses were taken into account. To deal with this extra memory and the time of occurrence of the responses made the buffer classifier more complex and took longer to run than the probabilistic classifier.

Figure 4.10 shows an example of the results obtained with this classifier. The results were obtained with the examples of the experiment on the figure 4.6 in section 4.4, that is:

$L_1= \{\vec{X}|\vec{X}=a^i b^j c^k d^l, i,j,k,l\geq 0\}$ and $L_2=U-L_1$

Here, in contrast with the probabilistic classifier in experiment 4.6, it was possible to distinguish between the different classes very well. Note that the buffer classifier has two properties in common with any model of temporal pattern recognition. Firstly, some memory of the input history is

required - in this case the buffer stores the order in which the input symbols occur in the sequences. Such a buffer does not limit the sequence length. Secondly, a function must be specified to combine the current memory (or the temporal context) with the current input to form a new temporal context.

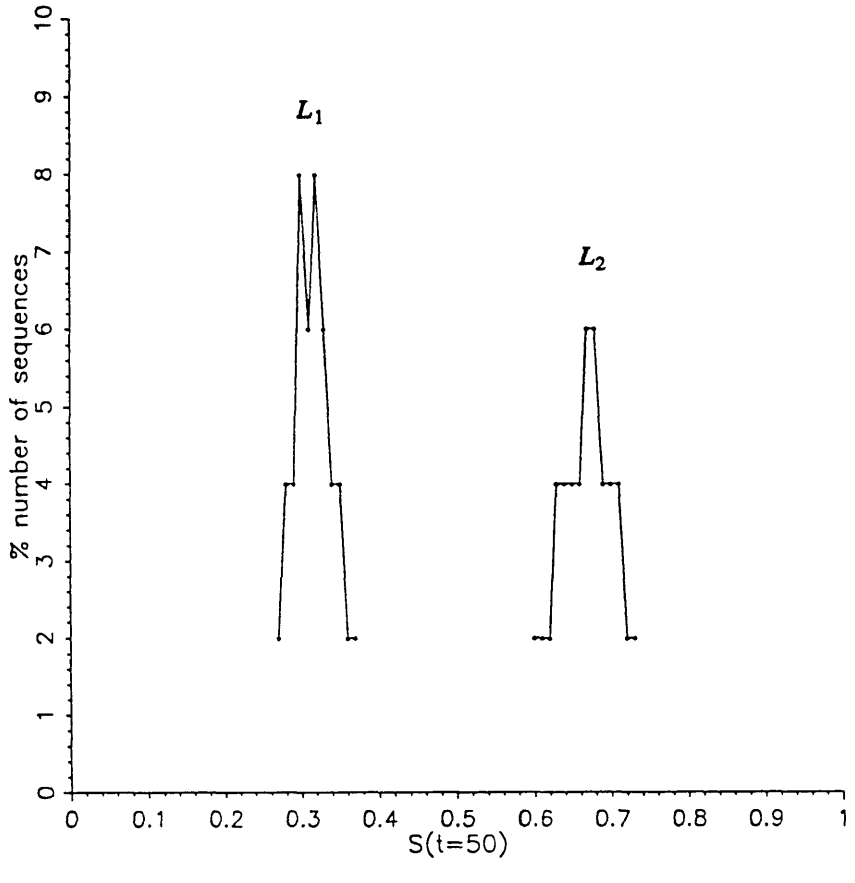


Figure 4.10 Buffer Classifier with $L_1 = \{\vec{X} | \vec{X} = a^i b^j c^k d^l, i,j,k,l \geq 0\}$ and $L_2 = U - L_1$.

4.7. FINAL STATE CLASSIFIER

Here ideas from formal language theory are used to recognise temporal patterns using neural networks. It is desirable to characterise the class of languages that can be recognised by a specific neural network.

One way to recognise temporal patterns using Neural Networks is through the final state of the network, which makes use of the way in which

finite automata work. As a result of this, instead of having a probabilistic calculation, a significant set of final states is required.

After the network has been trained, a finite-state structure is generated inside the network. The network is then fed with some patterns from the language to be recognised and the final state of the network for each sequence is stored. A sequence is deemed to have been recognised if the final state, after the test sequence has been passed, is within a certain Hamming distance of the final state for one of the training sequences.

$$S_k = \min \{dist(\delta(R_0,\vec{X}),R_F),\}/n;\ R_F \in F;\ k=1,2;$$

n is the number of bits in a symbol, R_0 is the initial state, R_F is a state belonging to the set of final states and F is the set of final states.

It is necessary to consider this Hamming distance because otherwise the generalisation with this classifier would be very small. Instead of calculating sums of probabilities, S_k is just the Hamming distance between the last state achieved with a sequence $\vec{X}$ and the closest state in the final state set. The Hamming distance is normalised in order to compare the results from this classifier with the results in the previous ones. In contrast with the two other classifiers, when $\vec{X}$ is in L_k, S_k should be close to 0 while when $\vec{X}$ is not in L_K, S_k should be close to 1.

This classifier was submitted to the same examples, as the probabilistic and buffer classifiers. Figure 4.11 shows the results for this classifier for the same problem as in figure 4.4. Figure 4.12 shows the results for the same problem as in figure 4.6.

As can be seen, in the examples above, the results with this classifier are worse than with the previous classifiers. In both experiments a complete separation between the classes was not achieved. The bad results come from the large generalisation. The network recognises all sequences in the language

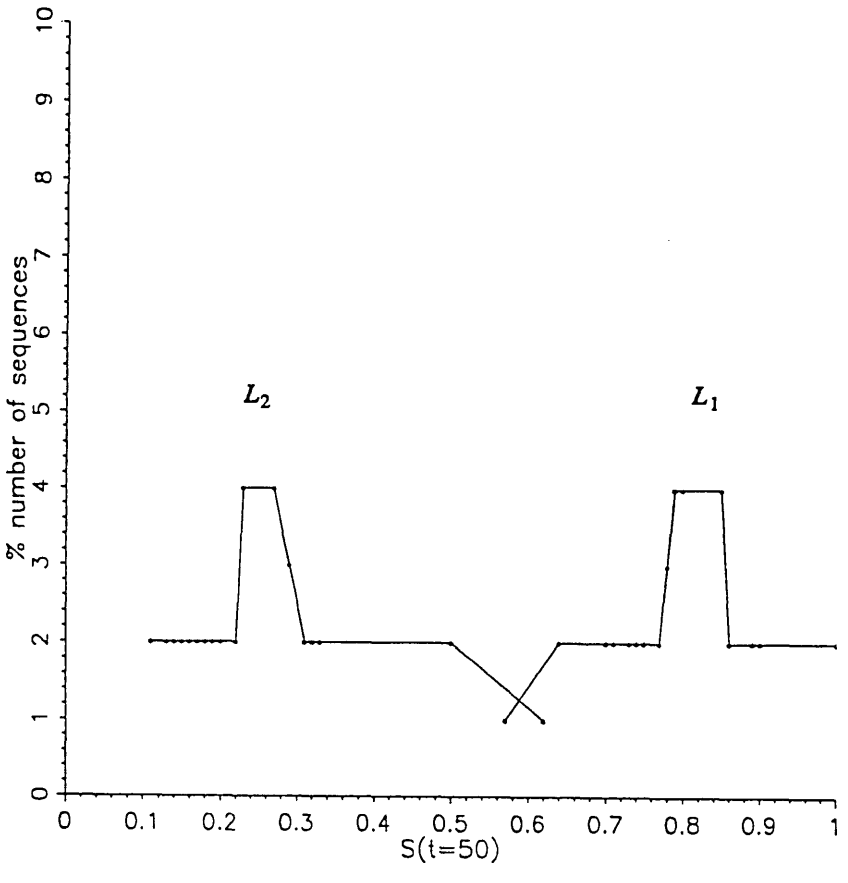


Figure 4.11 Final State Classifier with L_1=triangles and L_2=squares and circles.

it has been trained on but it also recognises some of the sequences which do not belong to the language (over-generalisation) because the Hamming distance allowed was too big. When a smaller Hamming distance were used some patterns belonging to the language were out of the generalisation set then the experiment with the bigger Hamming distance was chosen to be shown here. Also the network cannot recognise all the finite state languages because it is being trained only to predict the next symbol of the input sequence and there are finite state languages that do more than predict the next symbol(state).

With this classifier, the minimum difference in Hamming distance acceptable is decided upon depending on the problem to be solved. For instance, to solve the parity problem, the difference should be 0 because this

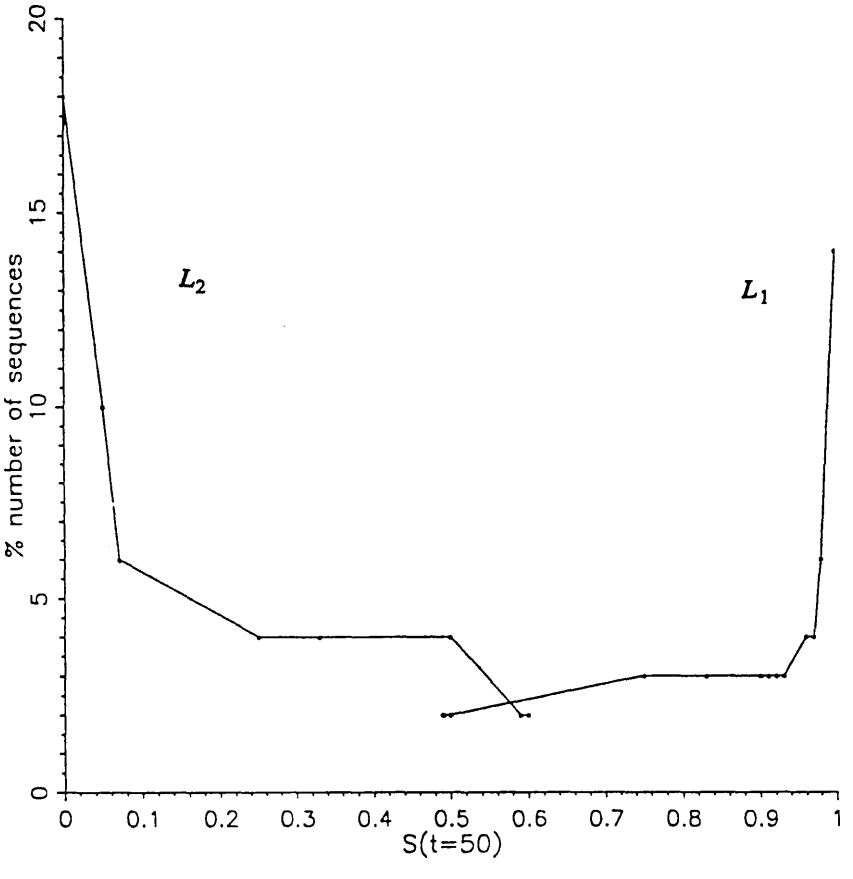


Figure 4.12 Final State Classifier with $L_1 = \{\tilde{X} | \tilde{X} = a^i b^j c^k d^l, i,j,k,l \geq 0\}$ and $L_2 = U - L_1$.

is a problem where a very small change in the input pattern causes a very big change in the output pattern (i.e. it changes the classification of the pattern). Another advantage of this classifier is that, it is very easy to know the total generalisation of the network by the regular expression. In the next chapter a method to calculate the total generalisation of a network is given based on automata theory.

4.8. CONCLUSION

A methodology has been presented for Temporal Pattern Recognition. The main ideas behind this methodology are: (1) the use of a feedback RAM-network (2) the use of different classifiers and (3) the study of the relation-

ship between stability and the recognition of patterns. The main strengths of the method are that: (1) the response of a RAM-network with feedback carries information about the order of appearance of its input patterns (2) the RAM-network is capable of recognising patterns independently of its initial states even in the presence of input distortions and (3) the generalisation emergent from these networks. Three different classifiers (probabilistic, buffer and final states) have been presented. The way that the output information of a network is processed makes a difference when working with different pattern recognition problems. Different classifiers are necessary to deal with different kinds of problems. The probabilistic classifier is efficient with the distinction of some sequences but if much sequential information needs to be processed the buffer classifier is more effective in distinguishing the sequences. On the other hand the buffer classifier is more complex than the probabilistic classifier. The goal with the final state classifier was different. With this classifier the idea was to measure the performance of the network as a finite state automaton.

The stability of a network was studied because it has an influence on its generalisation. Stability can be responsible for an increase in generalisation and can be controlled by the choice of a well defined class of functions performed by the neurons. Two different methods were used to control the stability: direct control and adaptive control. The number of feedback connections of the network was increased to make the network less stable. The results of the experiments were analysed using three different parameters: the size of state sets , the percentage of error recovery of input sequence and the amount of sequence distinction. Although stability increases generalisation, when a network is made stable by direct control, the classification made by the network is not always good. The number of feedback connections should

be low to make the network stable. But if the number of feedback connections is too small the network is not able to store the temporal context. With a high number of feedback connections only the adaptive control method produced recovery from input errors. With adaptive control the discrimination was better than with direct control. If the network is not stable, the state set is larger and the discrimination is poor.

If the stability control is made by the choice of the size of feedback connections, such size should be decided at the time the topology of the network is defined. It is not easy to change such size later on. In contrast, when the stability is controlled by the memory contents, the decision can be taken later on. As seen in the last paragraph the results from the study of stability facilitated the choice of parameters for the networks used in the discrimination experiments. For instance, a network should not have a big number of feedback connections when used for discrimination of sequences. In the case of a network of 32-RAMs with two input connections in each RAM, the number of feedback connections should not be greater than three. Actually, the best results were obtained with only two feedback connections.

CHAPTER 5

COMPUTABILITY OF LOGICAL NEURAL NETWORKS

5.1. INTRODUCTION

Since McCulloch and Pitts [McCulloch-Pitts, 43], there have been many studies of mathematical models of neural networks. Many concrete applications such as pattern recognition have been successfully tried. Many kinds of training techniques have been developed. However, there has been little theoretical research into the computability of such neural network models, with the exception of the work of Kleene [Kleene, 56] in the fifties.

In this chapter a new approach to pattern recognition, using PLN (Probabilistic Logic Node) networks, is introduced. With this method the network behaves as a probabilistic automaton. Using this method of recognition the computability of the network is increased beyond that of finite state machines. The class of languages that can be recognised by a logical neural network is compared with the classes of languages in Chomsky's hierarchy. In order to do this it is demonstrated that the computability of a PLN network and the computability of a probabilistic automaton are the same. To show that PLN networks and probabilistic automata have the same computability power it is only necessary to prove the following theorems:

Theorem 5.1 - Let G_w be a weighted regular grammar then there exists a PLN neural network that recognises $L(G_w)$ with some cut point λ, and

Theorem 5.2 - If a set of patterns L is recognised by a PLN neural network then this set can be generated by a weighted regular grammar G_w.

With theorem 5.1 a network to solve a specific stochastic problem can be designed. With theorem 5.2 the generalisation of a network and the functions

this network is able to compute can be determined. The two theorems together show the power of the network. Note that a probabilistic automaton can compute more functions than finite state automata, and once a proof that PLN networks have the power of probabilistic automata is given it is implied, then, that PLN networks are more powerful than RAM networks [Ludermir, 90c].

Additionally, having related logical neural networks to automata, some comments can be made. Firstly, logical neural networks can be trained, from a set of examples to solve a problem. Logical neural networks retain the emergent properties of traditional connectionist models [Aleksander, 83]. Secondly, as seen in chapter 4, logical neural networks can be made more powerful using some classifiers combined with the network.

The remainder of this chapter is divided into four sections. In the next section the new method of recognition is explained. In the third section an algorithm to transform any weighted regular grammar into a neural network is given, proving the first theorem. In the fourth section it is shown that a logical network can compute only weighted regular languages, thus proving the second theorem. The proofs of both theorems are algorithmic and are followed by examples. In the last section the advantages of such models are discussed and some conclusions are presented.

5.2. A PROBABILISTIC RECOGNITION METHOD

In this section a different way of achieving pattern recognition is introduced. This new method, using PLN networks, is based on the way probabilistic automata recognise patterns. In the method, the output of the network will consist of two parts. The first part is the recognition state of the network, which corresponds to a final state in a probabilistic automata. The second part

is the probability of the input pattern being recognised, as in the probabilistic automaton. A threshold, associated with the network, which has the function of doing the last step of the recognition of the system is introduced. Patterns will be recognised by the network if the probability associated with them is greater than the threshold. It will be shown that one of the advantages of this method is that the capacity of the network is increased. In the way PLN networks have been used to date, they could only recognise regular languages, whilst with the method proposed here they can recognise weighted regular languages.

Structure of the network

The network consists of several layers of probabilistic nodes. The functions performed by the probabilistic nodes are *COMPLEMENT*, *DELAY*, *p-AND*, *p-OR*. A *p-and* function is an operation which gives a value 1 if and only if all inputs $X_1, \ldots, X_n$ are equal to 1. The only difference between an *and* function and a *p-and* function is that in the memory position 2^n-1 of an *and* function there is a value 1 whilst in the memory position 2^n-1 of a *p-and* there is a value p. This value p is used to calculate the probability of a pattern being recognised by a network. A *p-or* function is an operation which gives a value 1 if at least one of $X_1, \ldots, X_n$ is equal to 1. As with the *and* and *p-and*, the only difference between an *or* function and a *p-or* function is that there are values p in the memories of the node which performs *p-or* whilst there are 1's in the memories of the *or* node. Each node can have one, two or several bits as input depending on the function the node performs. The nodes which perform *NOT* and *DELAY* functions have one input bit. The ones which perform *p-AND* functions and the ones which perform *p-OR* functions have two or more input bits. Every input pattern $\tilde{X}$ has a ψ character as its first input symbol. The ψ character only occurs once in each input pattern $\tilde{X}$.

Recognition algorithm

1. Choose an input pattern $\bar{X}=X_1X_2\cdots X_n$.
2. Feed the pattern to the network and calculate the probabilities of successful paths[1]
3. Calculate the total probability of the pattern $\bar{X}$.
4. See if the pattern is in the language to be recognised

As an example of the implementation of this algorithm a variable associated with every node of the network can be used. The purpose of this variable would be to store the probabilities of the path followed by the network until that node. For each input symbol X_i if the output of the node is equal to 1, to update the variable associated with this node. Otherwise put 0 in this variable. The updating step is better explained below. Suppose that a node k has n input bits, $i_1, i_2, \ldots, i_n$. Each of this input bits are the output of nodes $N_{i_1}, N_{i_2}, \ldots, N_{i_n}$. The variables associated with nodes $N_{i_1}, N_{i_2}, \ldots, N_{i_n}$ are $VA_{i_1}, VA_{i_2}, \ldots, VA_{i_n}$ and the variable associated with node k is VA_k. The updating of VA_k is done by the following procedure.

If function of node k is true (different from 0)

then begin

$VA_k := p_k$;

If function of node i_1 is true

then $VA_k := VA_k * VA_{i_1}$;

If function of node i_2 is true

then $VA_k := VA_k * VA_{i_2}$;

⋮

If function of node i_n is true

[1] A path is a sequences of states which the network went to when a pattern $\bar{X}$ was submitted as the input. A successful path is a path for which the last state of the path is in the set of the final states of the network for the language to be recognised.

then $VA_k := VA_k * VA_{i_n}$;

else $VA_k := 0$.

where p_k is the probability associated with node k.

After the last symbol of the input pattern is fed to network, only the variables associated with the successful path will have their values different from 0. To calculate the total probability of the pattern $\bar{X}$, the values of all the variables are summed. To know if the pattern $\bar{X}$ belongs to the language L the total probability of this pattern is compared with the threshold.

This algorithm was only designed to show that the computability of PLN networks could be increased and the algorithm can be improved in the future. This algorithm was not used in many applications. One drawback of PLN networks in relation to pattern recognition is that there is no deterministic decision as to whether a pattern $\bar{X}$ belongs to the language recognised by a given PLN network. The way this algorithm works makes the network generate the same answer whenever a pattern $\bar{X}$ is submitted to the network. Although the structure of the network described in the beginning of this section is very specific, this algorithm will work with any other structure of the network.

5.3. FROM GRAMMARS TO NEURAL NETWORKS

In this section an algorithm for transforming any weighted regular grammar into a PLN neural network is given. The way in which the grammar is transformed into the neural network is such that all the properties of the grammar are preserved and it is possible to infer the grammar from the PLN neural network generated. Networks constructed with four kinds of nodes: *p-and*, *p-or*, *complement* and *delay* nodes will be considered. Any function of n inputs can be represented by an expression involving only the operations of

AND, OR and COMPLEMENT [Booth, 71] then these networks will be able to represent all functions of n inputs.

As in the recognition algorithm it is assumed that all sequences have an initial symbol ψ. For example, suppose that a language L has three patterns $\vec{X}$=ABC, $\vec{Y}$=BCA and $\vec{Z}$=ACB. To be used in the next theorem these pattern need to be transformed in $\vec{X}=\psi$ABC, $\vec{Y}=\psi$BCA and $\vec{Z}=\psi$ACB. This ψ symbol tells the network that a new pattern is being submitted to the network and it occurs only in the beginning of each sequence. There is no loss of generality in the supposition of each sequence beginning with a ψ symbol since it is possible to define a node to deal with this symbol. Every PLN network will contain exactly one *delay* node to deal with the symbol ψ. The input of this *delay* node is the symbol ψ. For example, in figure 5.1 below, the only *delay* node is the one which deal with the symbol ψ. N' will denote the network N without the *delay* node whose the input is the symbol ψ. The input terminal which connects the ψ symbol to the *delay* node will be called the ψ-input of a network. In figure 5.1, for instance, the ψ-input is the input connection of the *delay* node. With networks N's, the ψ-input of the network is the output of the *delay* node (the one which deals with the ψ symbol).

Each non-terminal symbol (the symbols belonging to the set V_N) occurs at most twice in the left-hand side of the production rules. That is, grammars with productions like $S \rightarrow w_1 | \cdots | w_n$, $n>2$, for example, will not be considered. The argument can be easily, but tediously, extended for those cases. Also, it is possible to rewrite grammars where such non-terminal symbol occurs more than twice in the left-hand side of the production rules in a way that all non-terminal symbols in this situation will occur twice at most.

Theorem 5.1. *Let $G_w=(V_N,V_T,P_w,S)$ be a weighted regular grammar. Then there exists a PLN neural network that recognises $L(G_w)$ with some cut-point λ.*

Proof.

The proof of this theorem is based on the complexity of the production rules. Transformation of the production rules, in PLN networks, will be started by the most simplest production rule.

Case 1. The production rule is of the form $S_1 \rightarrow w\ (p)$, $w \in V_T^*$ and $S_1 \in V_N$.

a) $S_1 \rightarrow w\ (p)$, $w \in V_T^+$. If the only production rule is $S_1 \rightarrow w\ (p)$ it denotes a set containing only one pattern and it is recognised by the network constructed from a k+1 input *p-and* node whose j^{th} input is the j^{th} network input if X_j, the j^{th} symbol of the sequence (pattern), is 1 , otherwise is the output of a *complement* node whose input is the j^{th} network input X_j. The last input of the *p-and* node is the output of a *delay* node which has ψ as input. The output of the network N_S is the output of the *p-and* node. This network is shown in figure 5.1 below where for $j=1$, $X_j=X_1=1$ then the first input of the *p-and* node comes directly from the first input of the network and for $j=2$, $X_j=X_2=0$ then the second input of the *p-and* node is the output of a *complement* node which the input is the second input of the network.

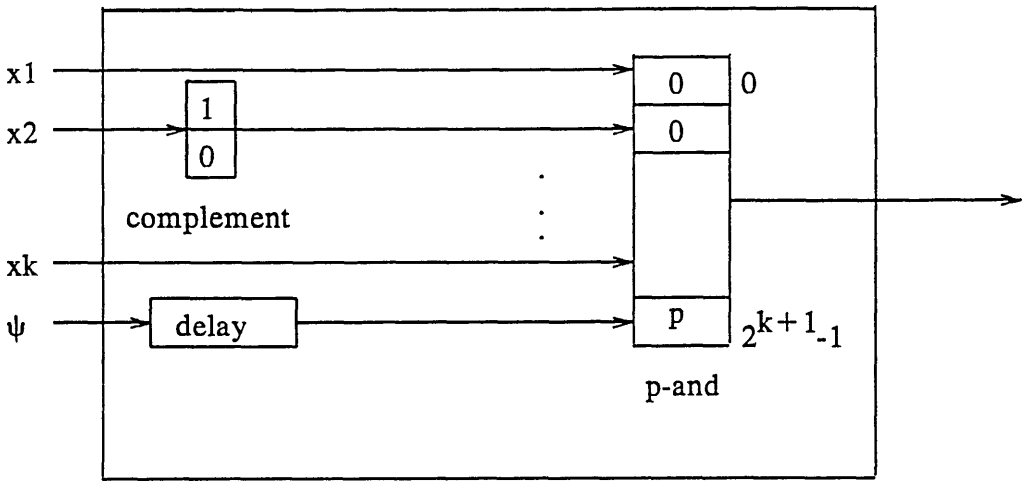
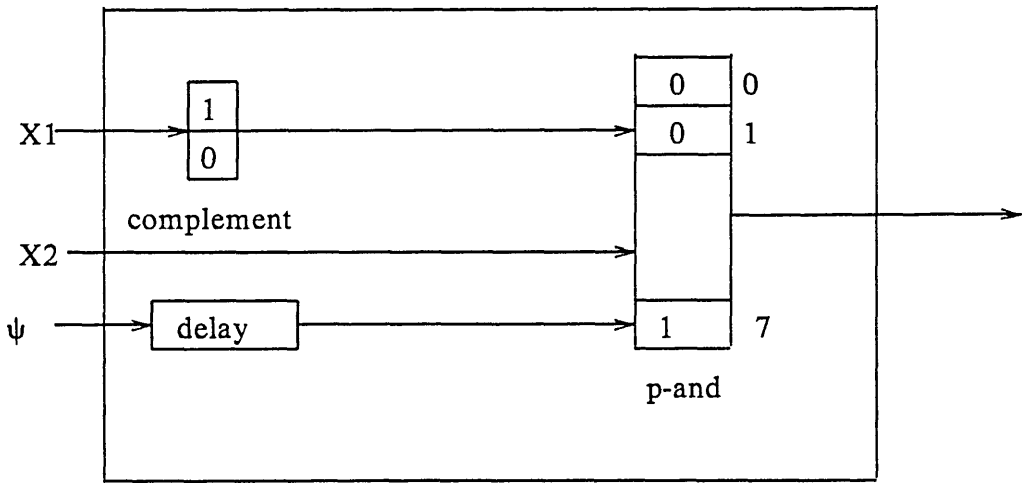


Figure 5.1 A network for $S_1 \rightarrow w$ (p).

Below an example of a grammar which generates only one pattern is given.

$G_w=(V_N,V_T,P_w,S)$ where $V_N=\{S\}$, $V_T=\{0,1\}$ and P_w: $\{S \rightarrow 01(p=1)\}$ The only pattern generated by this grammar is 01.



The only pattern recognised by this network is 01.

b) $S \rightarrow \epsilon$. If the only rule is $S \rightarrow \epsilon$ it denotes the empty set. There are several networks which are able to recognise the empty set. The one chosen here is a

network constructed from a k+2-input *p-and* node, a *delay* node for the input ψ and a *complement* node as shown in figure 5.2 below:

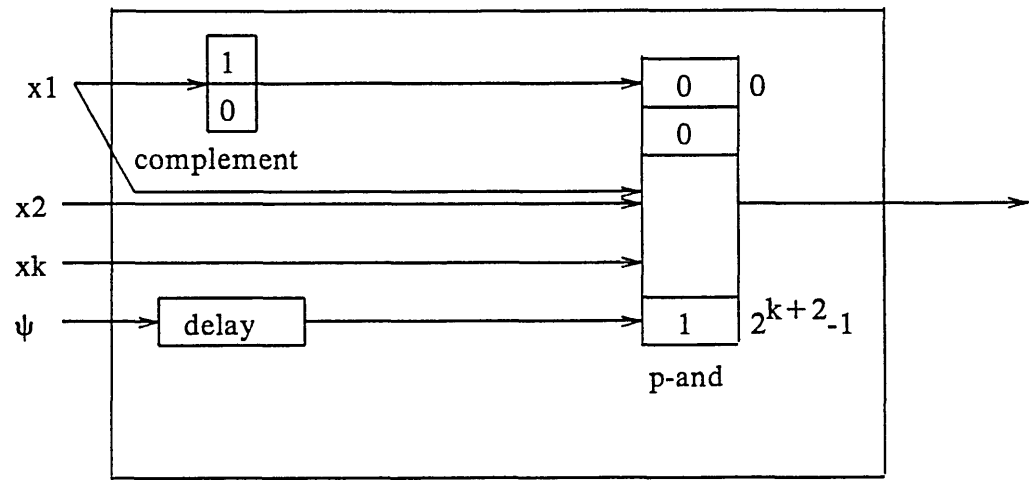
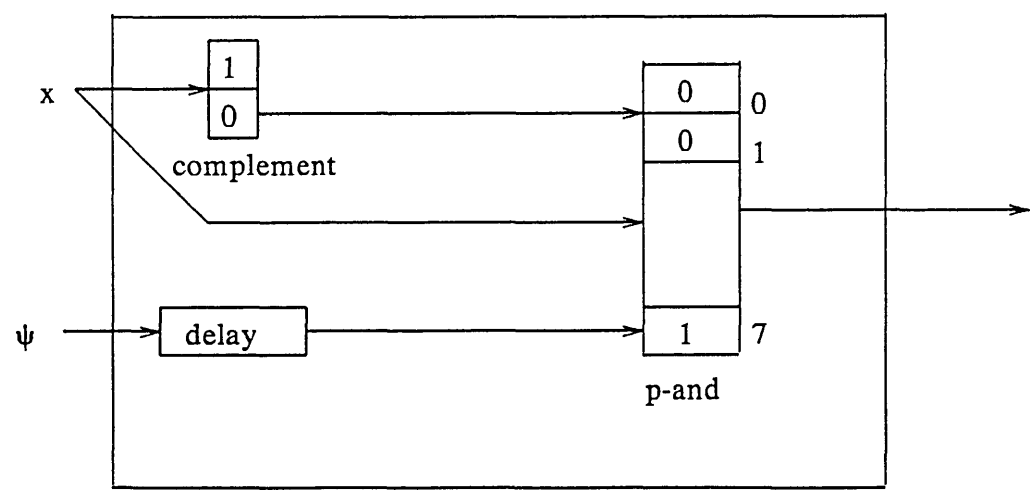


Figure 5.2 A network $S \rightarrow \epsilon$.

The network was chosen to be constructed in this way in order to make it similar with the other networks of this proof.

Below, an example of a grammar which generates only the empty pattern is given. $G_w=(V_N,V_T,P_w,S)$ where $V_N=\{S\}$, $V_T=\{\epsilon\}$ and P_w: $\{S \rightarrow \epsilon\}$



The only pattern recognised by this network is the empty pattern ϵ.

Case 2. The production rule is of the form $S_i \rightarrow wS_j(p)$.

a) $S_i \rightarrow wS_j(p)$ with $i<j$. Suppose that w and S_j are generated by a grammar with production rules as in case 1. Thus there are networks N_w and N_{S_j} which recognises w and S_j. Now, the network N_{S_i} is constructed from N'_w, N'_{S_j} and two *delay* nodes as in figure 5.3 below. The output of N'_w is the input of of a *delay* node 2. The ψ-input of the network N'_{S_j} is the output of *delay* node 2. The output of N_{S_i} is the output of N'_{S_j}.

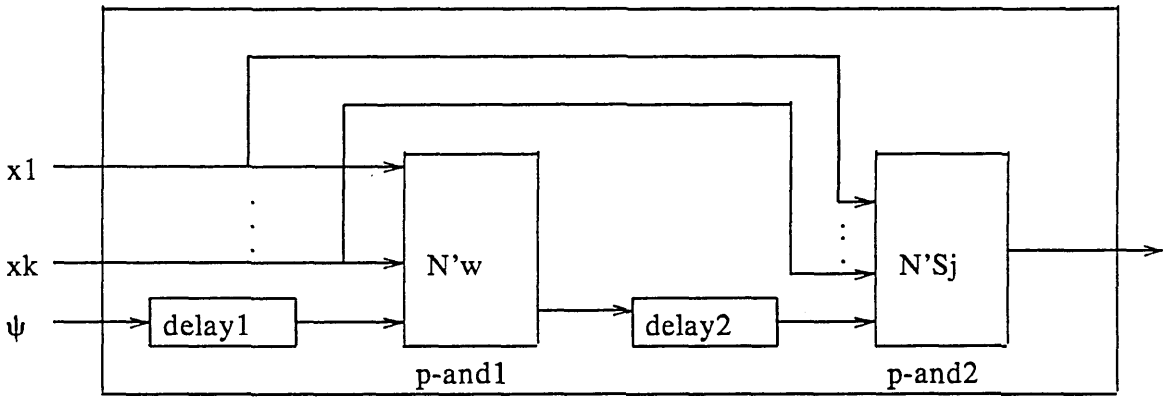


Figure 5.3 N_S, a network for $S_i \rightarrow wS_j(p)$.

b) $S_i \rightarrow wS_j(p)$ with $i=j$. Let N_w be as in case (a) above and construct N_{S_i} from N'_w, two *delay* nodes and a 2-input *p-or* node as in figure 5.4 below. The output of N'_w is the input of *delay* node 2. The output of *delay* node 2 is one of the inputs of the *p-or* node. The other input of the *p-or* node is the output of *delay* node 1. The input of *delay* node 1 is the symbol ψ. The output of the *p-or* node is ψ-input of N'_w. Note that there is no output in this network S_i. To make sense every time a recursive production rule happens, there should be an output for this recursive production rule. The recursive production rule with output will be dealt with in case 3 of this theorem.

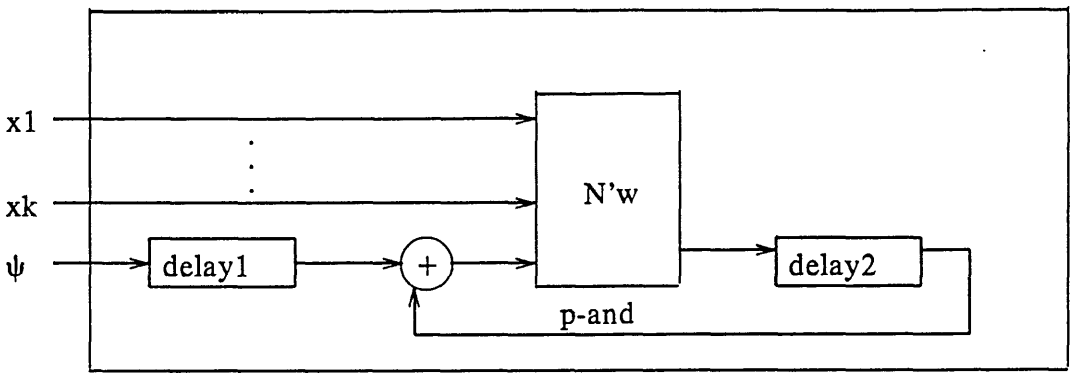


Figure 5.4 N_S, a network for $S_i \rightarrow wS_j(p)$ with $i = j$.

c) $S_i \rightarrow wS_j(p)$ with $i > j$. This case can be considered similar to case (b) above. Let N_w and N_{S_j} be as in case (b) above and construct N_{S_i} from N'_w, N'_{S_j}, two *delay* nodes and a 2-input *p-or* node as in figure 5.5 below. The output of N'_w is the input of the *delay* node. The output of the *delay* node is one of the inputs of the *p-or* node. The other input of the *p-or* node is the output of the *delay* node associated with the original ψ-input of N_{S_j}. The output of the *p-or* node is the ψ-input of N'_{S_j}. The output of N'_{S_j} and the ψ-input of N'_w remain unchangeable. It is possible to have a loop between N_{S_j} and N_w, but this is not generally the case..

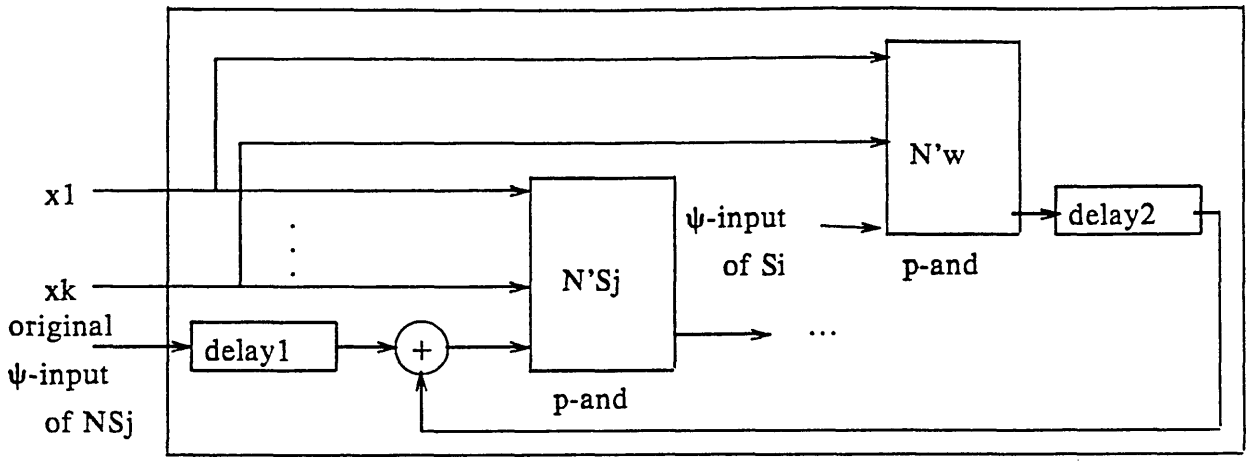


Figure 5.5 N_S, a network for $S_i \rightarrow wS_j(p)$ with $i > j$.

Case 3. The production rule is of the form $S_i \rightarrow w_1 S_j(p_1) | S_i \rightarrow w_2 S_k(p_2)$.

a) $S_i \rightarrow w_1 S_j(p_1) | S_i \rightarrow w_2 S_k(p_2)$ with $i < j,k$. Let N_{w_1}, N_{w_2}, N_{S_j}, N_{S_k} be as in case 2 and constructed N_{S_i} from N'_{w_1}, N'_{w_2}, N'_{S_j}, N'_{S_k}, three *delay* nodes and a 2-input *p-or* as in figure 5.6 below. The input of the *delay* node 1 is ψ. The output of the *delay* node 1 is the ψ-input of the networks N'_{w_1} and N'_{w_2}. The output of N'_{w_1} is the input of the *delay* node 2. The output of *delay* node 2 is the ψ-input of N'_{S_j}. The output of N'_{S_j} is one of the input of the 2-input *p-or* node. The other input of the *p-or* node is the output of N'_{S_k}. The output of N'_{w_2} is the input of *delay* node 3. The output of the *delay* node 3 is the ψ-input of N'_{S_k} and the output of the *p-or* node is the output of S_i. Note that w_1 and w_2, or S_j and S_k, or w_1 and S_k, or S_j and w_2, or only w_1, or only w_2, or only S_j, or only S_k need not exist and this would not change the methodology of the construction of the network.

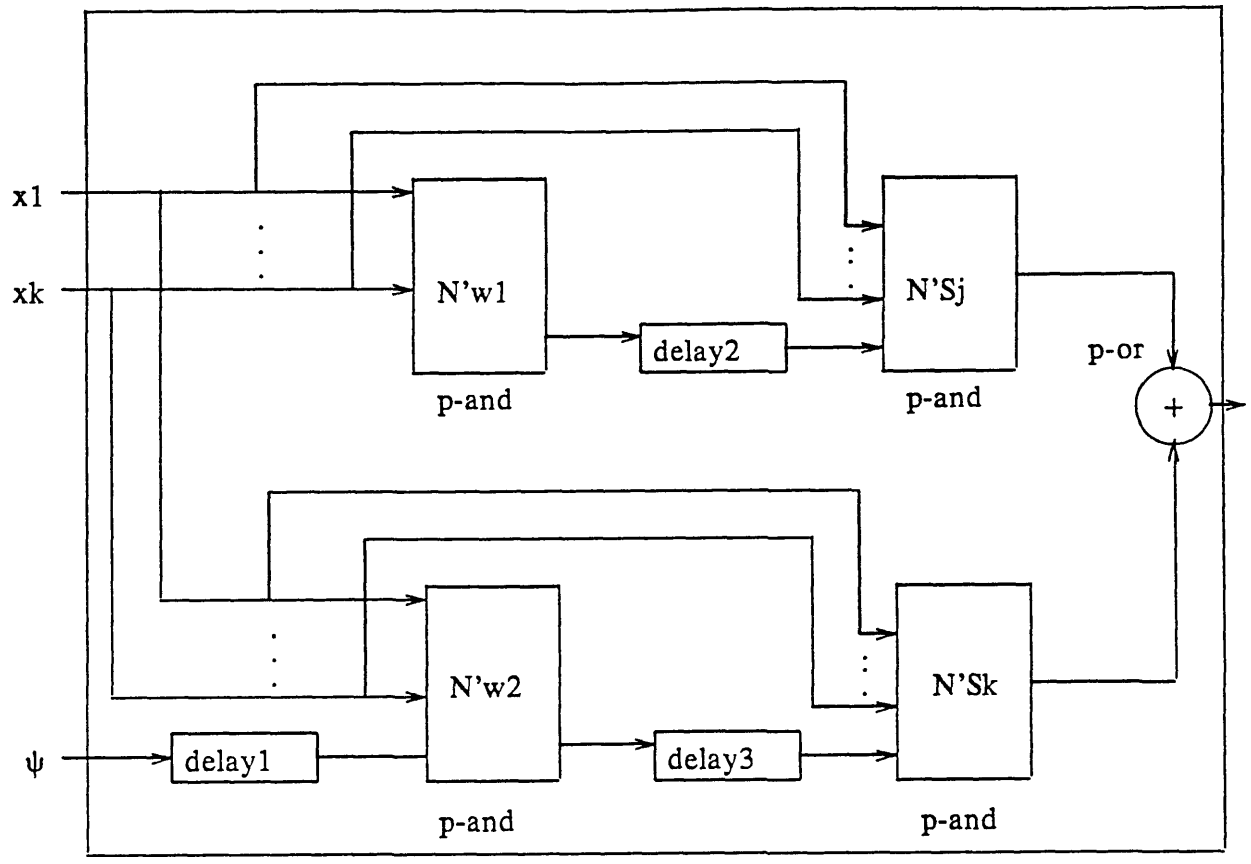


Figure 5.6 N_S, a network for $S_i \rightarrow w_1 S_j(p_1) | S_i \rightarrow w_2 S_k(p_2)$ with $i < j,k$.

b) $S_i \rightarrow w_1 S_j(p_1) | S_i \rightarrow w_2 S_k(p_2)$ with $i = j$ and $i < k$. Let N_{w_1}, N_{w_2}, N_{S_k} be as in case 2 and constructed N_{S_i} from N'_{w_1}, N'_{w_2}, N'_{S_k}, three *delay* nodes and a 2-input *p-or* node as in figure 5.7 below. The input of *delay* node 1 is ψ. The output of *delay* node 1 is one of the input of 2-input *p-or* node. The other input of the *p-or* node is the output of the *delay* node 2. The output of the *p-or* node is the ψ-input of the network N'_{w_1} and N'_{w_2}. The output of N'_{w_1} is the input of *delay* node 2. The output of N'_{w_2} is the input of the *delay* node 3. The output of *delay* node 3 is the ψ-input of N'_{S_k} and the output of N'_{S_k} is the output of N_{S_i}. As in case (a) above, S_k need not exist and this would not change the methodology of the construction of the network.

If $i=j$ and $i<j$ above the same construction can be applied, only changing S_k by S_j.

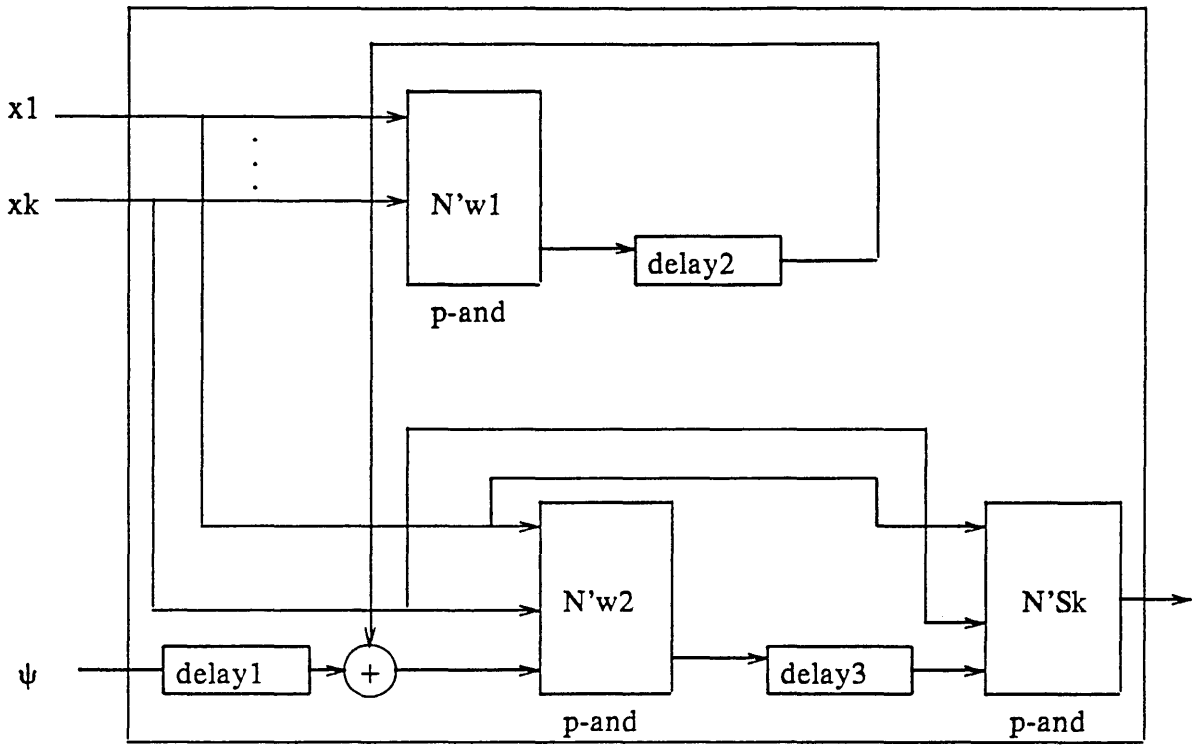


Figure 5.7 N_S, a network for $S_i \rightarrow w_1 S_j(p_1) | S_i \rightarrow w_2 S_k(p_2)$ with $i=j$ and $i<k$.

c) $S_i \rightarrow w_1 S_j(p_1) | S_i \rightarrow w_2 S_k(p_2)$ with $i=j=k$. Here both production rules are recursive. There is no way out from such a loop. Although such production rules are not expected to happen in grammars, a solution for them will be given for completeness. This case is very close to case (b) above. The construction in (b) to S_j will be applied also to S_k here. This is shown in figure 5.8 below.

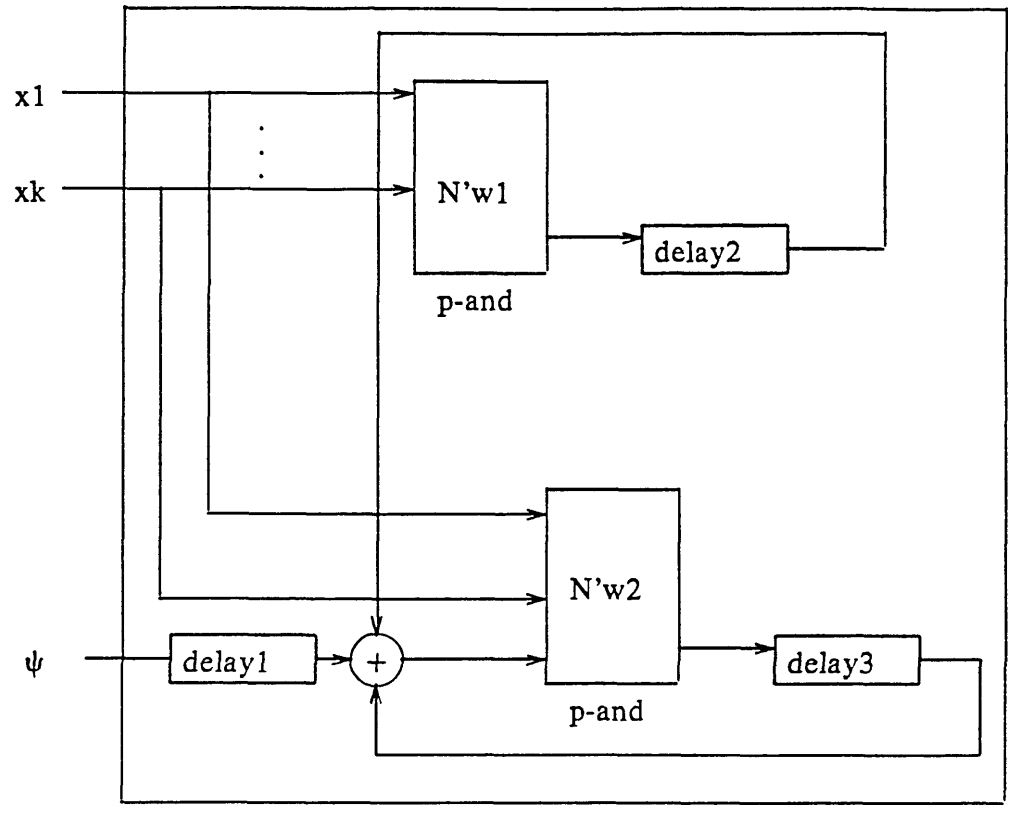


Figure 5.8 N_S, a network for $S_i \rightarrow w_1 S_j(p_1) | S_i \rightarrow w_2 S_k(p_2)$ with $i = j = k$.

d) $S_i \rightarrow w_1 S_j(p_1) | S_i \rightarrow w_2 S_k(p_2)$ with $i > j$ and $i < k$. The first part of this case ($S_i \rightarrow w_1 S_j(p_1)$ with $i > j$) is equivalent to case 2 (c) and the second part ($S_i \rightarrow w_2 S_k(p_2)$ with $i < k$) is equivalent to case 2 (a). When considering both production rules ($S_i \rightarrow w_1 S_j(p_1)$ and $S_i \rightarrow w_2 S_k(p_2)$) it is necessary to put them together as in case 3 (b). This is shown in figure 5.9 below.

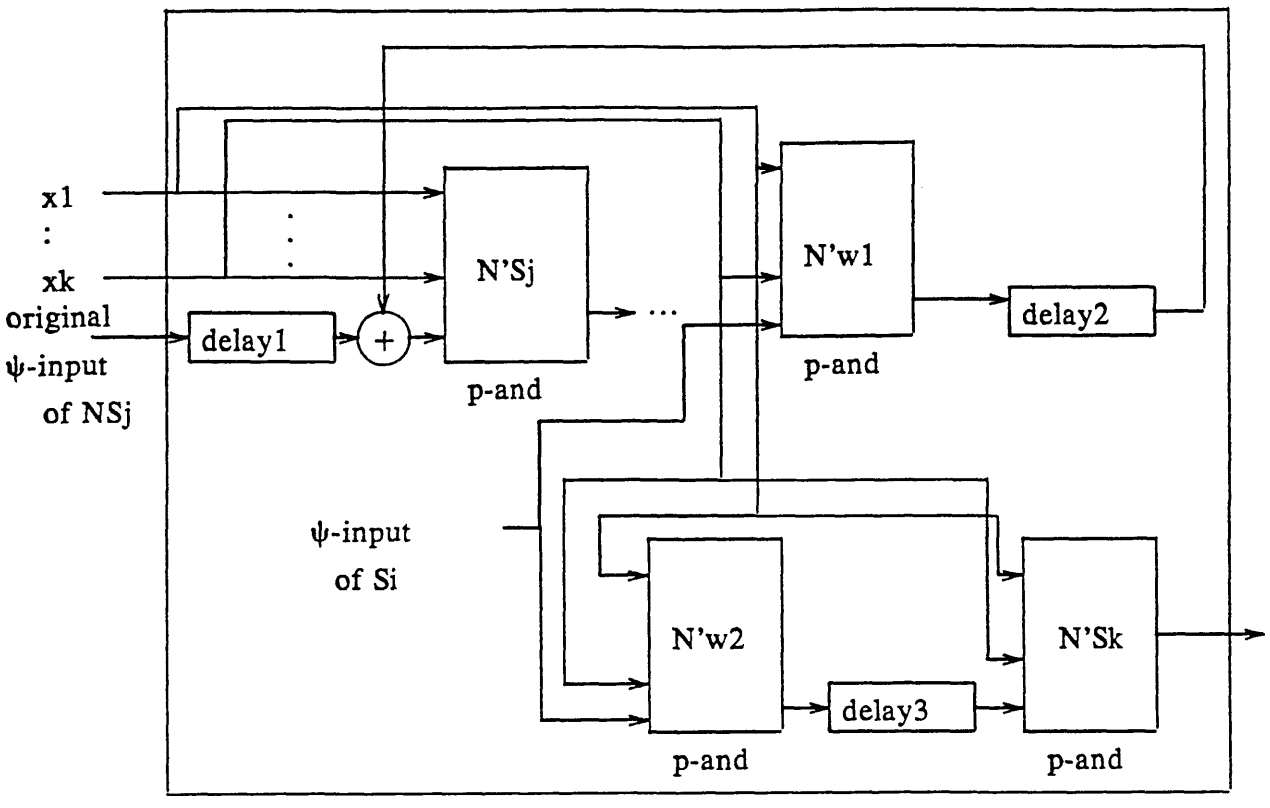
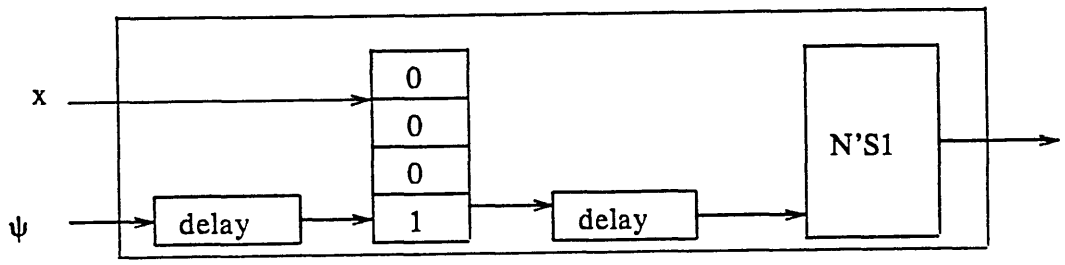


Figure 5.9 N_S, a network for $S_i \rightarrow w_1 S_j(p_1) | S_i \rightarrow w_2 S_k(p_2)$ with $i > j$ and $i < k$.

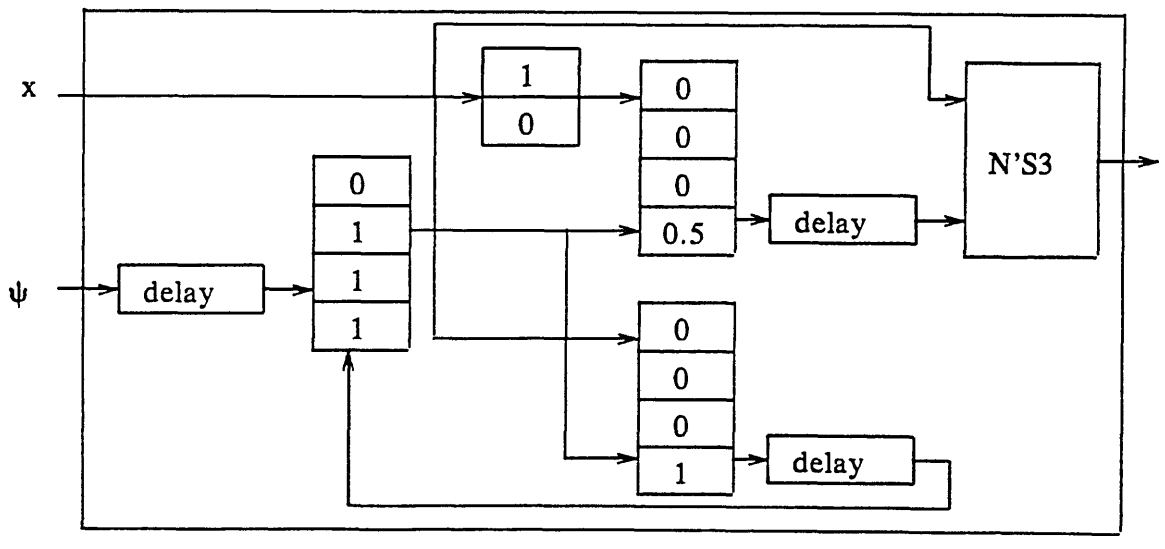
If $i < j$ and $i > k$ above the same construction can be applied only changing S_k by S_j.

If $i > j$ and $i > k$ above the same construction can be applied for both S_j and S_k.

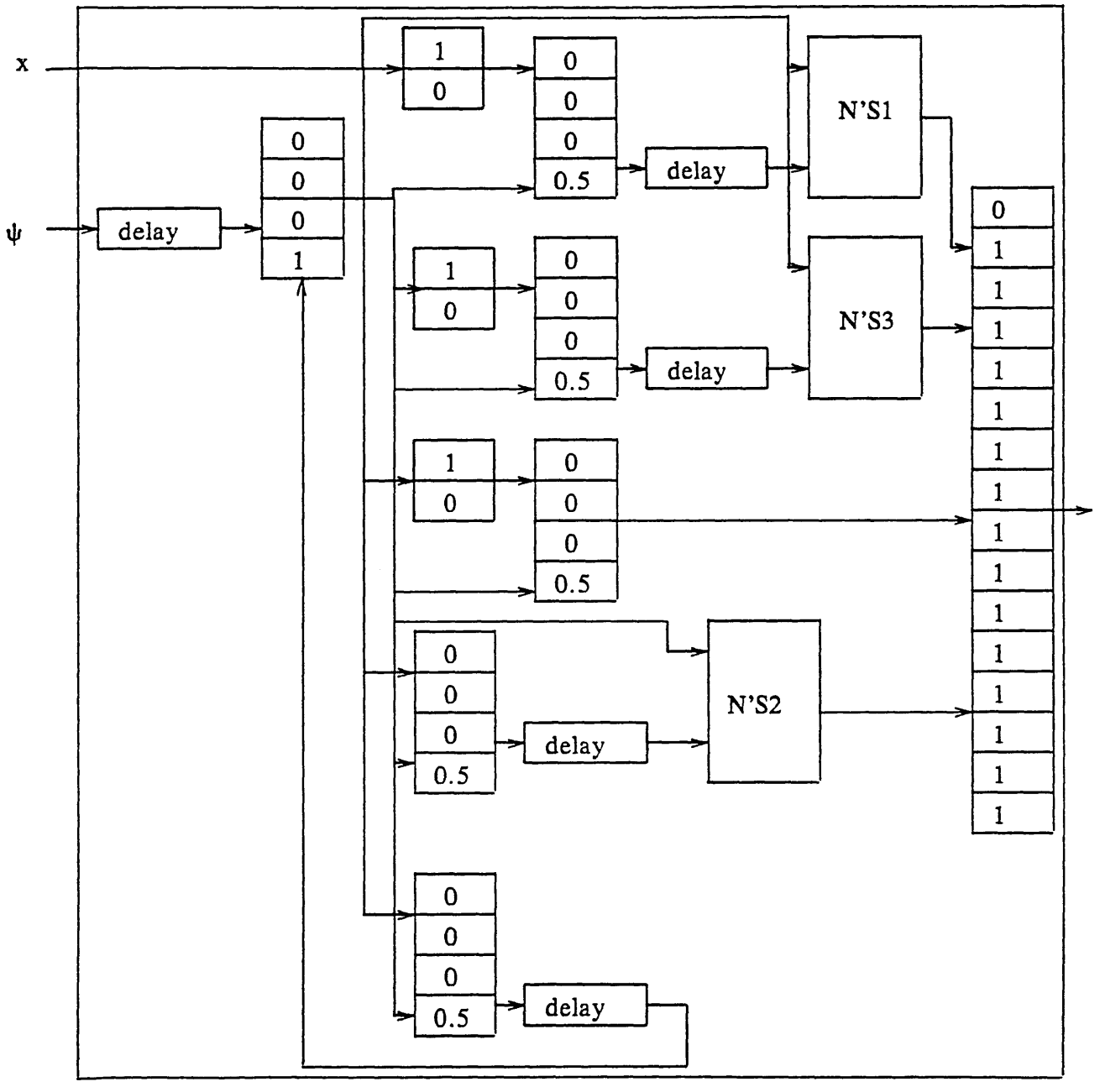
Now, an example of a conversion of a weighted regular grammar into a PLN neural network is going to be considered. Given the weighted regular grammar G_w in the example of definition 3.3 in chapter 3 (page 45), in what follows, networks which recognise each set of production rules for every symbol in V_N of G_w are shown and in the last figure a network which recognises $L(G_w)$ is shown.



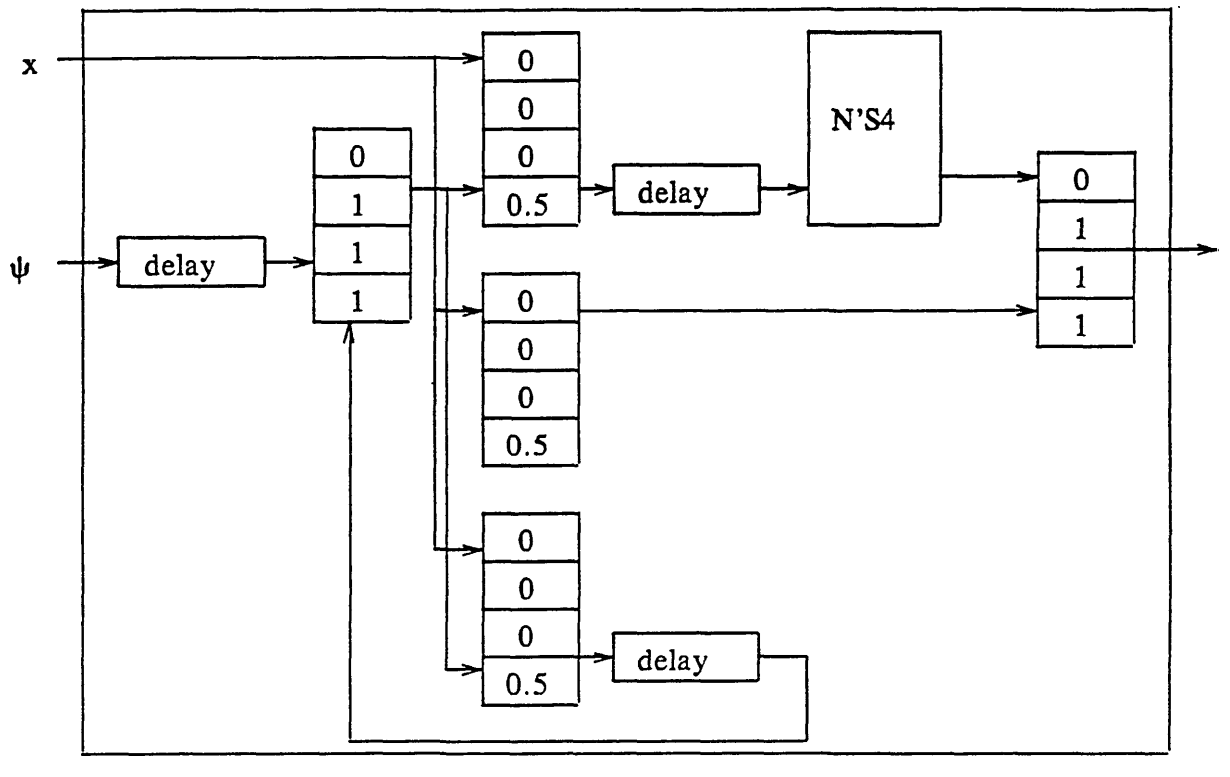
NET S



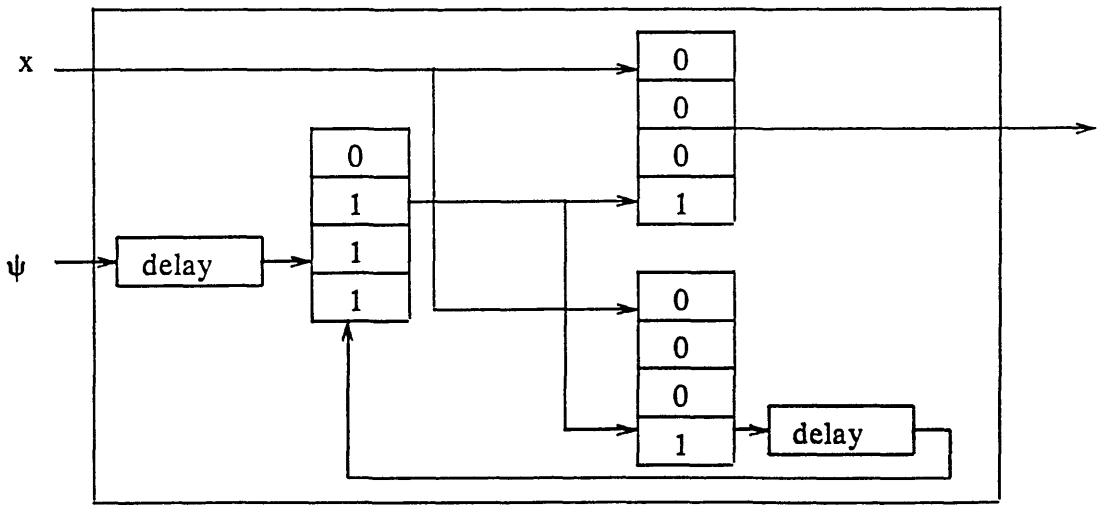
NET S2


x
ψ
delay
1
0
0
0
0
0.5
delay
N'S1
0
0
0
1
1
0
0
0
0.5
delay
N'S3
1
0
0
0
0
0.5
1
0
0
0
0.5
0
0
0
0.5
delay
N'S2
0
0
0
0.5
delay
0
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1

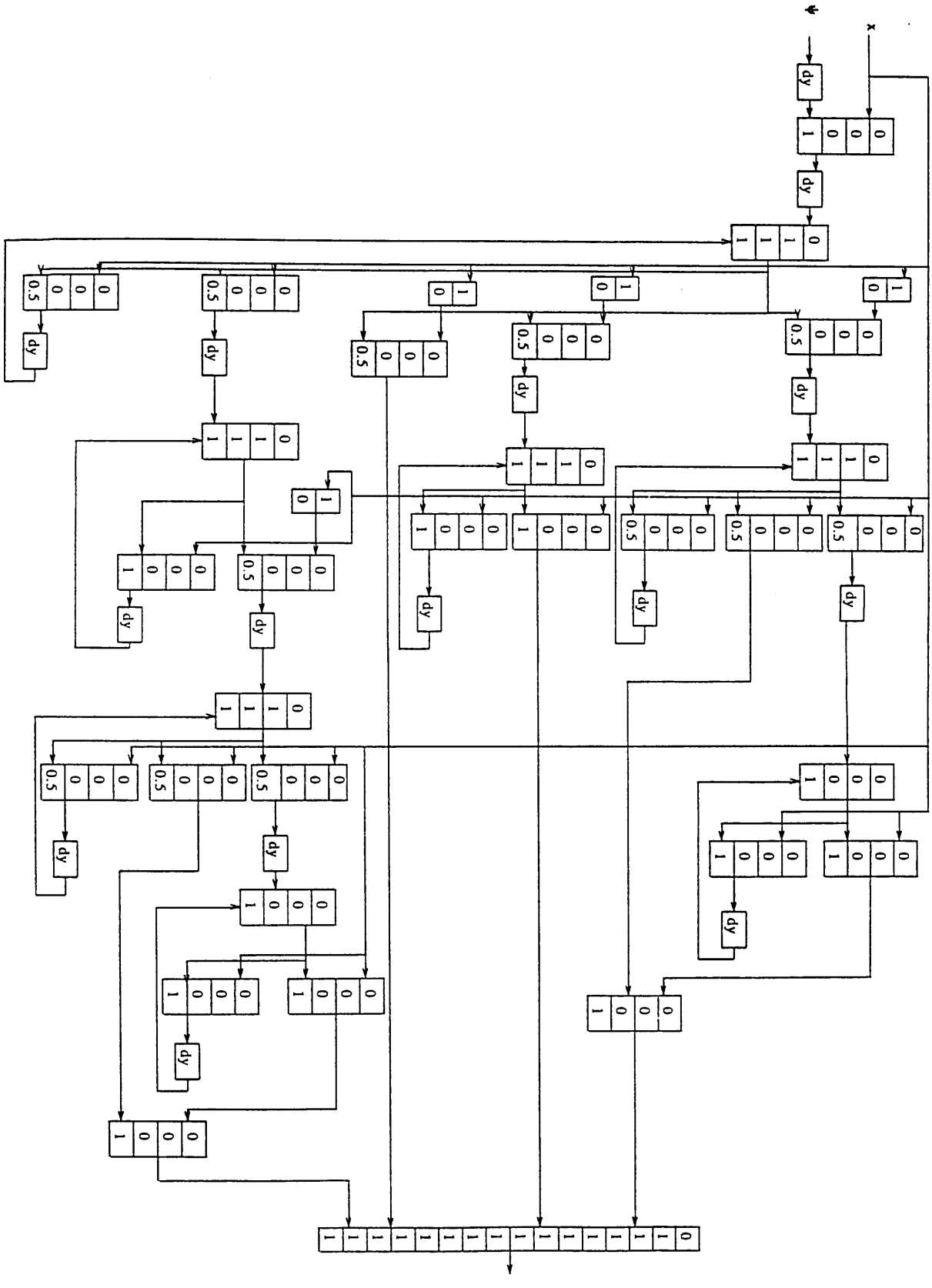
NET S1


x
ψ
delay
0
1
1
1
0
0
0
0.5
delay
N'S4
0
1
1
1
0
0
0
0.5
0
0
0
0.5
delay

NET S3


x
ψ
delay
0
1
1
1
0
0
0
1
0
0
0
1
delay

NET S4


x
dy
1 0 0 0
1 1 1 0
0.5 0 0 0
1 0
0 1
1 0 0 0
0.5 0 0 0
1 1 1 0
1 0 0 0
0.5 0 0 0
1 1 1 0
1 0 0 0
0.5 0 0 0
1 1 1 0
1 0 0 0
0.5 0 0 0
1 0 0 0
1 0 0 0
0.5 0 0 0
0.5 0 0 0
0.5 0 0 0
1 0 0 0
1 0 0 0
1 0 0 0
1 0 0 0
1 0 0 0
1 0 0 0
1 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0

NET G_w

Regular grammars can be transformed in RAM networks using the same algorithm; but where the neuron stores 0, 1 and not probabilities. The proof has been omitted and is implied by the following facts: firstly, as regular grammars and RAM neurons are special cases of weighted regular grammars and PLN neurons respectively, it is clear that the algorithm will work for them. Secondly, it is well known that RAM networks are finite state machines and so they are able to recognise finite state languages. And lastly, the method for such proof is similar to the one for PLN networks. But as regular grammars are not probabilistic, instead of putting probabilities in the memory of the neurons, only the normal *and*, *or*, *not* and *delay* nodes are necessary. No weighted regular grammar of languages other than regular ones can be implemented in RAM networks, since RAM networks are finite state machines and finite state machines can only recognise regular languages.

Although the algorithm presented in this section gives a complete structure - the network and its memory contents - the language recognised by the network can be changed in three ways. This is particularly useful when the exact grammar of the language to be recognised is not known, only an approximation. Two ways of changing the language to be recognised involve training the network generated by the algorithm. The third, only involves changing the threshold. The class recognised by a probabilistic automaton may change accordingly with the change of the cut-point λ [Rabin, 63]. This is also true for logical networks. The languages recognised by a network when only the threshold is changed, are related to each other. The higher the value of the threshold the fewer the elements of the language will be recognised by the network. That is, $L_1 \supset L_2 \supset \ldots \supset L_n$ when $\lambda_1 < \lambda_2 < \cdots < \lambda_n$. If the threshold is small there are more restrictions in the path followed in the network: every time a new symbol, X_i, is submitted to the network the value of the probabil-

ity of the pattern $p(\vec{X})$ will decrease or will not change, but it will never increase. The first way of training will change only the probability stored in the memory of the nodes. When the probabilistic state transition of the probabilistic automaton is slightly changed, the probabilistic automaton will, sometimes, recognise a different language [Rabin, 63]. Unfortunately, this is not true all of the time. There are some sufficient conditions for stability in probabilistic automata and there are cases in which stability is not possible. The general problem of stability is still unsolved, which means that if the changes generated by the training algorithm in the state transition of the network are small there is no guarantee that the training changes the language recognised by the network. The training algorithm in this case can be very simple. Given a pattern $\vec{X}$, if $\vec{X} \in L$ then to reward the network (decrease the probabilities of the transitions the network went to with $\vec{X}$) otherwise then to punish the network (increase the probabilities of the transitions the network went to with $\vec{X}$). A training algorithm which allows changes in any memory position in the network is also possible. In this case the function computed by the network can change completely. The generation of the network by the algorithm described here is useful as an initial set up of the network.

5.4. FROM NEURAL NETWORKS TO GRAMMARS

In this section it is demonstrated that every set of patterns recognised by a PLN neural network can be generated by some weighted regular grammar. Again, as the main goal here is theoretical, optimisation is not a concern. Actually, this theorem is necessary only to show that the relationship between weighted regular languages and PLN networks is an *if then if* relation. This algorithm can be used to determine the total generalisation of a network after the network has been trained. This method of calculating the total

generalisation is more efficient than many others, for example: submitting patterns to the network to see if they are recognised by the net. It is also better than going through the whole network in order to calculate the generalisation. The method derived from the theorem gives also the probability of recognition for each pattern in the class recognised by the network. At the end of this section an example of this theorem with a PLN network will be given.

Theorem 5.2. *Any set of patterns L which is recognised by a PLN neural network can be generated by a weighted regular grammar.*

Proof.

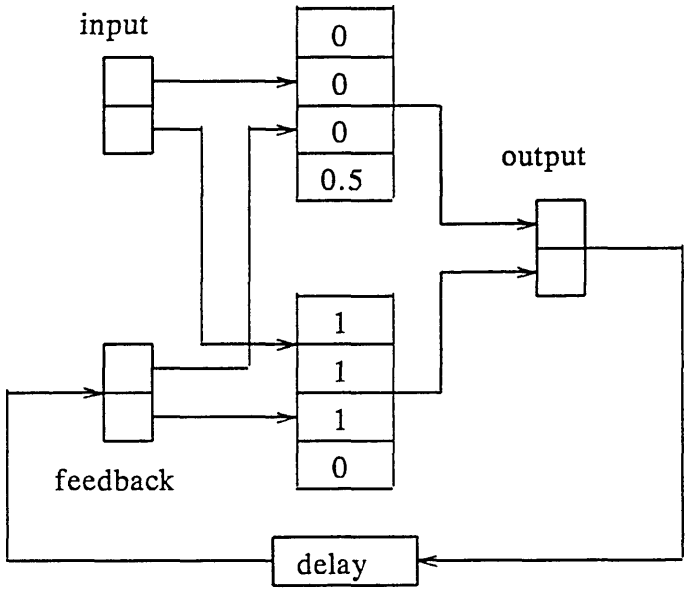
Let N be the PLN neural network which recognises only the set of patterns L and let $G_w=(V_N,V_T,P_w,q_0)$ be the grammar which can generate only and all patterns of L. Suppose there is an initial state q_0 of N into which the feeding all patterns of L will start, and suppose now that q_0 is not a final state. Then there is a production rule $S_i \to aS_j(p)$ whenever the feeding of the symbol a to the network in state S_i causes the network to enter state S_j with probability p, and also $S_i \to a(p)$ whenever the feeding of the symbol a to the network in state S_i takes the network to a final state with probability p. In the same way, there is a set of production rules such that $S_i \overset{*}{\Rightarrow} wS_j(p)$, whenever the feeding of the pattern w to the network in state S_i causes the network to enter state S_j with probability p. If w is accepted by N then, S_i is q_0 and S_j is a final state. Hence $L(N)=L(G)$.

Now let q_0 be in the set of final states, then ϵ is in L. Note that the grammar defined above generates $L-\{\epsilon\}$. G_w can be modified by adding a new start symbol S with productions $S \to q_0(p_1) \mid \epsilon\ (p_2)$.

Note that the method used to prove this theorem is similar to that of proving that a recogniser (automaton) and a generator (grammar) are dealing

with the same language in formal language theory.

Now an example of a transformation of a PLN neural network to a weighted regular grammar is given. The PLN neural network is given in the figure below.



Suppose that the states of the network are S_1=00, S_2=01, S_3=10, and S_4=11; the initial state is S_1; The final state set is $F=\{S_2\}$; and the inputs are a=00, and b=11. Then the grammar G_w will be definied as follows:

$G_w=(V_N,V_T,P_w,S_1)$ where $V_N=\{S_1,S_2,S_3,S_4\}$, $V_T=\{a,b\}$ and

P_w: (1) $S_1 \rightarrow aS_2$ $(p=1)$

(2) $S_1 \rightarrow bS_2$ $(p=1)$

(3) $S_2 \rightarrow aS_2$ $(p=1)$

(4) $S_2 \rightarrow bS_1$ $(p=1)$

(5) $S_3 \rightarrow aS_2$ $(p=1)$

(6) $S_3 \rightarrow bS_2$ $(p=0.5)$

(7) $S_3 \rightarrow bS_4$ $(p=0.5)$

(8) $S_4 \rightarrow aS_2$ $(p=1)$

(9) $S_4 \rightarrow bS_1$ $(p=0.5)$

(10) $S_4 \rightarrow bS_3$ $(p=0.5)$

All languages recognised by a RAM network can be generated by some regular language. The same procedure of theorem 5.2 can be used to show this well known result. The proof of this fact has been omitted for the same reasons the proof that regular grammars can be transformed in RAM networks was omitted. The advantages of PLN networks, mentioned in the beginning of this section, are also true in the case of RAM networks.

5.5. CONCLUSIONS

A new method of recognition of patterns with PLN networks, based on the way probabilistic automaton recognises patterns using cut-point, has been introduced. This method increases the number of functions which can be computed with PLN networks with respect to conventional methods used to date. With conventional methods PLN networks have the computability power of a finite state machine whilst, with the method introduced in this chapter, PLN networks can compute all weighted regular languages. Having studied the relationship between logical (PLN and RAM) neural networks and automata (deterministic and probabilistic) and having analysed the computability of such logical neural networks, a formal characterisation of the languages recognised by these logical neural networks has been obtained. Algorithms to transform logical neural networks into automata and vice versa have been provided. It may be worth noting that it is still possible to increase the power of these networks with the addition of classifiers combined with the network, as was done with RAM networks in the preceding chapter.

Although logical neural networks have the same computability as probabilistic automata it must be remembered that such networks learn how to recognise a set by training while with automata the set of productions rules have to be known in order to construct the network. In this work, the

networks were not trained to recognise weighted regular languages from examples. A pushdown automaton network for the task of grammatical inference as in [Giles et al., 90] could have been developed but this was not the goal here. In their work, Giles et al. were able to solve some simple context-free problems while here it was shown that it is possible to do much more than this with logical networks. For finite state languages, a network to recognise the language from a set of examples can always be designed. For instance, the learning algorithm developed by Porat and Feldman [Porat-Feldman, 88] to obtain the automaton for the set of examples can be used. Their algorithm will always learn the minimum state deterministic automaton for any finite state language which is presented to the learning algorithm in strict lexicographic order. The algorithm in section 5.3 can be used, then, to transform any regular grammar into a RAM net.

Based on the stability property of probabilistic automaton, discussed in chapter 3, ones knows that even slight changes in the probabilistic state transition δ can, in some cases, force the automaton recognise a different language. Of course, big changes in the probabilistic state transition δ will make the automaton recognise a different language in all cases. Although, the main goal here is to show the computability power of logical networks, it is possible to use the procedure of theorem 5.1 to create the structure of a network and then train this network to recognise a language which the grammar to generate this language is not known. This structure can be trained in two different ways as explained in section 5.3. The language recognised by the network can also be modified only by changing the value of the threshold.

As repetited concluded in the body of this chapter it has been shown that PLN networks can compute more functions than RAM networks. It is interesting to note that, in passing from a minimal deterministic automaton to

an equivalent probabilistic automaton sometimes it is possible to save states [Rabin, 63]. The same kind of results were observed experimentally with RAM and PLN networks. It is important to draw attention to the fact that PLN neural networks are more powerful, that is, can compute more functions, than networks composed of McCulloch- Pitts neurons. This can be concluded by looking at Kleene's results for McCulloch-Pitts networks and the results of theorems 5.1 and 5.2 in this chapter.

By the computability power of McCulloch-Pitts networks one can show that such networks can compute many of the hard learning problem (hard learning problems were defined and discussed in section 2.2 of this thesis). Kleene [Kleene, 59] showed that McCulloch-Pitts networks are equal to finite state machines. Parity, for instance is a finite state problem, therefore McCulloch-Pitts networks can solve parity. McCulloch-Pitts networks can solve easily all hard learning problems which are regular. There are hard learning problems which are not regular, as for example symmetry. In this case a more sophisticated learning algorithm is necessary. If, when Minsky and Papert presented their argument against perceptrons [Minsky-Papert, 69], an analysis of the problem had been made through the computability of McCulloch-Pitts networks one would have realised that what was necessary was to change the topology of such networks. Of course, this would not have easily revealed a way to train the network, but at least the research at the time could have moved in the direction of analysing new topologies and looking for training algorithms.

CHAPTER 6

CONCLUSIONS

The first section of this chapter summarises the contributions of the research work described in the previous chapters and the last section discusses possibilities for future work based on the results achieved in this thesis.

6.1. SUMMARY OF ACHIEVEMENTS

The primary aim of the work reported in this thesis was to investigate the ability of logical neural networks to deal with temporal pattern recognition tasks. With this in mind, three systems using RAM networks with feedback were developed and used for temporal tasks. A new method of recognition using PLN nodes was proposed which was shown to have increased the computability power when compared with logical neural networks. It was demonstrated that the computability power of PLN networks and probabilistic automata are the same. The main contributions of this thesis are summarised in greater detail in the following paragraphs.

CLASSIFICATION OF SEQUENCES

This thesis presented three different systems for recognising sequences. All these systems were based on RAM networks with feedback, where the feedback causes information about the order of appearance of input patterns to be carried. The networks in all the systems were trained to anticipate their input symbols. This was denoted as $D(t)=X(t+\alpha)$, such that $R(t)=X(t+\alpha)$ might be obtained during the test phase. This training strategy was chosen with the objective of recognising sequences using the states of the network. It

was shown that it also made the network stable with respect to small changes in input sequences.

It was seen that the output of the network did not have enough information to be able to deal efficiently with temporal pattern recognition. So, the computability power of the systems was increased by the introduction of different classifiers combined with the network. Such different classifiers were considered to cope with different ways in which the output information of a network is processed when working with different problems. The three different classifiers used were the probabilistic, the buffer and the final state classifiers. Below a short description of these classifiers and the results achieved with each one of them are given.

PROBABILISTIC CLASSIFIER

This classifier was based on probabilities of occurrence of patterns in each of the two classes, L_1 and L_2, to be distinguished. The frequency with which output symbol symbol R_i of the network occurs for sequences of both classes L_1 and L_2 was used in the discrimination process. In other words, the probability $p(\vec{X} \in L_k/\vec{R})$ that the input sequence $\vec{X}$ belongs to the language L_k, given that the response $\vec{R}$ occurred when $\vec{X}$ was fed into the network, was used in this classifier. $p(\vec{X} \in L_k/\vec{R})$ changed randomly near 1 for sequences $\vec{X}$ in L_k and near 0 for sequences $\vec{X}$ not in L_k. The continuous average $S_k(t)$ of $p(\vec{X} \in L_k/\vec{R})$ was used for considering $p(\vec{X} \in L_k/\vec{R})$ for all symbols in the same sequence $\vec{X}$. $S_k(t)$ tended to 1 for all sequences belonging to the language L_k, while $S_k(t)$ tended for 0 for all sequences not belonging to the language L_k when the distinction between the two classes was possible. This classifier was effective at distinguishing different geometric forms, the first class of problems that were dealt with. However, this classifier was not efficient in the dis-

tinction of different regular languages, the second class of problems dealt with. One of the reasons for the unsatisfactory results with the regular languages was that in such languages the position of the symbols in the sequences (and not occurrence frequencies) was crucial for determining whether the sequence to be classified as belongs to a language or not. This meant that a great deal of sequential information had to be stored by the network, which was not possible with this classifier.

BUFFER CLASSIFIER

This classifier was also based on the probability $p(\bar{X} \in L_k/\bar{R})$ that the input sequence $\bar{X}$ belongs to the language L_k, given that the response $\bar{R}$ occurred when $\bar{X}$ was fed into the network. This classifier was designed to be more powerful than the probabilistic classifier, once it was found that the probabilistic classifier was not able to distinguish between regular languages: powerful in the sense that this classifier was able to process more sequential information, that is, it used the information about the order of occurrence of the input symbols in the input sequences also. The extra power of this classifier was introduced by the use of a buffer, to save more sequential information. Such a buffer was needed because although symbols occurred at different positions in different classes, they could occur with the same probability in both. Hence, it was necessary to store the time of occurrence of the states. This classifier was submitted to the same experiments as the probabilistic classifier. Better results were achieved with the buffer classifier than with the probabilistic classifier, but the system was more complex.

FINAL STATE CLASSIFIER

This classifier was based on ideas from formal language theory. The final state of the network was used in the discrimination of the two classes and instead of calculating probabilities, a significant set of final states was required. A sequence was recognised (accepted) if, after being fed into the network, its final state was within a certain Hamming distance from the final state of one of the training sequences. This classifier was submitted to the same examples as the probabilistic and buffer classifiers. The results with this classifier were worse than with the previous classifiers. The network could not recognise all the final state languages because it was trained only to predict the next symbol of the input sequence and there are finite state languages that do more than predict the next symbol. The goal with this classifier though was different: it was to measure the performance of the network as a finite state automaton. Two advantages of the final state classifier were:

1) the minimum difference in Hamming distance acceptable could be set depending on the problem to be solved. For instance, to solve the parity problem, the difference should be 0 because this is a problem where a very small change in the input pattern causes a very big change in the output pattern (changes the class of the pattern) and

2) it is very easy to predict the total generalisation of the network by the regular expressions.

INFLUENCE OF THE STABILITY PROPERTY IN THE CLASSIFICATION OF SEQUENCES

The stability of the network with relation to errors in the input sequence was studied because it is central to the discrimination process. Two different methods were used to control the stability: direct control, where the function

of each neuron was selected at random in such a way that the same DL (the difference between the number of 1's and 0's in memory) could be met for all neurons; and adaptive control, where the distribution of 0's and 1's was caused by training. The number of feedback connections also influenced the stability of the network and it was increased in order to make the network less stable. Three different parameters were used to analyse the experimental results when changing the size of the feedback connection and the way of controlling neuron memory contents: the size of the state sets; the percentage of error recovery in input sequences; and the amount of sequence distinction. The state sets of each class should neither be large nor should there be many common states. The percentage of recoveries from error in an input sequence influenced the size of the state set in each language.

Experiments were done with the networks of 32 RAMs, with 2 input connections and the number of feedback connections varied from 2 to 6, that is, SDNN(32,2,2), SDNN(32,2,3), SDNN(32,2,4), SDNN(32,2,5) and SDNN(32,2,6). The results confirmed the fact that the number of feedback connections should be low to make the network stable, i.e., not more than 3 connections in case of the experiments in this thesis. The best results were achieved with 2 feedback connections. Although stability in the network increases generalisation, when a network is made stable by direct control, the classification made by the network is not always good.

DETERMINISTIC RECOGNITION ALGORITHM USING PLN NODE

A new recognition algorithm using PLN nodes was proposed. This new method increases the number of functions which can be computed with logical neural networks with respect to conventional methods used to date. With conventional methods PLN networks have the computability power of a finite

state machine whilst, with the method introduced in this thesis, PLN networks can compute all weighted regular languages. This algorithm was based on the way probabilistic automata recognise patterns using cut-points. The algorithm makes use of the probabilistic information stored in the memory of the node and the output of the network consisted of two parts. The first part is the recognition state of the network and the second part is the probability of the input pattern being recognised. Patterns are recognised by the network if the probability associated with them is greater than the threshold associated with the language. This algorithm not only increases the computability power of the network but also it is more appropriate to pattern recognition because it gives a deterministic decision as to whether a pattern $\vec{X}$ belongs to the language recognised by a given network or not.

EQUIVALENCE BETWEEN LOGICAL NETWORKS AND PROBABILISTIC AUTOMATA

The equivalence of the computability power of a PLN network and a probabilistic automaton have been demonstrated. To show that a PLN network and a probabilistic automaton have the same computability power it was only necessary to prove the two following theorems:

- (Theorem 5.1) Let G_w be a weighted regular grammar then there exists a PLN neural network that recognises $L(G_w)$ with some cut point λ, and
- (Theorem 5.2) If a set of patterns L is recognised by a PLN neural network then this set can be generated by a weighted regular grammar G_w.

From the proof of theorem 5.1 an algorithm was designed which, given any weighted regular language, could construct a neural network that could recognise it, thus a network can be designed to solve any specific stochastic problem. From the proof of theorem 5.2 another algorithm was designed to gen-

erate a weighted regular grammar for the patterns recognised by a given network. With this theorem the generalisation of a network and the functions it is able to compute can be determined. The two theorems together showed the power of the network. Since probabilistic automata can compute more than finite state machines, it follows that PLN networks can compute more than RAM networks. Many neural problems, for instance, natural language understanding, make use of context free grammars, therefore it is important to be able to implement in neural networks at least some of the context-free grammars. The methods used to prove that PLN networks and probabilistic automata have the same computability power can be used straightforwardly to prove that RAM networks and finite state automata have the same computability power although this is evident from the start. Three different ways to change the language recognised by the network generated by the algorithm from theorem 5.1 were given. This is particularly useful when the exact grammar for the language to be recognised is not known, only an approximation. Two of the ways of changing the language involve training the network generated by the algorithm and the third involves changing the threshold only.

6.2. IMPROVEMENTS AND SUGGESTIONS FOR FUTURE WORK

The study of the relationship between automata theory and neural networks is at an early stage. Much work must to be done before the results from automata theory can be exhaustively applied to neural networks. However, automata theory have been used to come to conclusions about the computability power of logical networks and to show that logical networks can compute the same functions as probabilistic automata.

There are several interesting points that were not examined in this thesis as for instance the complexity of learning in such logical networks. Although,

it was shown that logical networks could compute weighted regular languages, no mention was made of how long a network needs to learn a certain weighted regular language. That is, how complex is the learning algorithm for such networks. A study of the complexity of learning was made by Judd for unidirectional feed-forward networks of MCP nodes [Judd, 90]. Although his results are quite independent of the details in the nodes themselves he did not tackle the problem for feedback networks. The complexity of the learning problem in its general form (the learning of any function) is too difficult to solve. One way of making the problem easier is by considering particular constraints on the learning problem. There are fast learning algorithms for cases where the network is of a very restricted design, or where the data to be learned are very simple.

A way of increasing the class of languages that can be recognised by logical networks was given. Now it is possible to deal with all the weighted regular languages using logical networks when before it was only possible to deal with regular languages. There are weighted regular languages which are context-free, context-sensitive and even unrestricted (type 0) languages. The possibility of dealing with weighted regular language by itself is, then, a very good result but the practical shortcomings of such a result was not really investigated. Clearly, this investigation needs to be done soon.

The problem of training was not a concern of this work even though in chapter 5 some considerations were made to the way the networks generated by theorem 5.1 could be trained. More work needs to be done in order to specify the properties a training set needs to have for a successful training of the network. Also, different training algorithm and strategies should be developed in the future to make use of the information provided by the node.

Since it has been shown, in this thesis, which functions can be recognised

by logical networks, a study can be made of the possibility of increasing the power of these networks even further with the addition of classifiers combined with the network, as was done with RAM networks in chapter 4.

As most of the studies of neural networks are based on simulation results, simulations should be done using different temporal problems to compare the experimental results with the results from other models. Also, the recognition algorithm proposed in chapter 5 can be made more efficient, for example, instead of having a variable associated to every node of the network with the purpose of storing the probability of the the path followed until that node, to use the memory of the own node to store such probability. This algorithm should be used in further experiments and its performance should be measured.

The distant goal of 'neural networkers' is to understand how to store, retrieve, and process data in neural networks; ultimately to characterise the types of data that need to be stored, to know how best to represent them, and to see how to design such machines that accomplish it with the greatest engineering ease. It is hoped that the results of this thesis have contributed to this quest.

REFERENCES

[Allinson-Johnson, 89] Allinson, N. M.; Brown, M. T. & Johnson, M. J. (1989) $\{0,1\}^n$ space self-organising feature maps – extensions and hardware implementations. *Proc. 1st IEE International Conference on Artificial Neural Networks*, London, pp. 261-264.

[Aleksander-Hanna, 76] Aleksander, I. & Hanna, F.K. (1976) Automata Theory: An Enginnering Approach. Edward Arnold, London.

[Aleksander-Mamdani, 68] Aleksander, I. & Mamdani, E.H. (1968) Micro-circuit Learning Nets: Improved recognition by means of pattern feed-back. *Electronics Letters*, **4**(20), pp.425-426.

[Aleksander-Mamdani, 70] Aleksander, I. & Mamdani, E.H. (1970) Universal Sequential Logic Elements. *Electronics Letters*, **6**(25), pp. 801-802.

[Aleksander-Morton, 90] Aleksander, I. & Morton, H. (1990) An Introduction to Neural Computing. Chapman and Hall, London.

[Aleksander-Stonham, 79] Aleksander, I.; Stonham, T. (1979) Guide to pattern recognition using random-access memories. *IEE J. Computers and Digital Tech.*, **2**(1): 29-40.

[Aleksander et al., 84] Aleksander, I.; Thomas, W.V. & Bowden, P.A. (1984) WISARD, a radical step forward in image recognition. *Sensor Review* **4**(3), pp 120-124.

[Aleksander-Wilson, 85] Aleksander, I.; Dobree-Wilson, M. (1985) Adaptive windows for image processing. *IEE Proceedings*, **132E**(5): 233-245.

[Aleksander, 66] Aleksander, I. (1966) Self-adaptive Universal Logic Circuits. *Electronics Letters*, **2**, pp. 231.

[Aleksander, 71] Aleksander, I. (1971) Microcircuit Learning Computers. Mills & Boon Ltd., London.

[Aleksander, 83] Aleksander, I. (1983) Emergent Intelligent Properties of Progressively Structured Pattern Recognition Nets. *Pattern Recognition Letters*, **1**, pp. 375-384.

[Aleksander, 88] Aleksander, I. (1988) Logical connectionist systems. In, *Neural Computers* (eds. R. Eckmiller, C. von der Malsburg). Springer-Verlag, Berlin, pp. 189-197.

[Aleksander, 90] Aleksander, I. (1990) Ideal neurons for neural computers. In, *Parallel Processing in Neural Systems and Computers* (eds. R. Eckmiller, G. Hartmann, G. Hauske). North-Holland, Amsterdam, pp. 225-228.

[Al-Alawi-Stonham, 89] Al-Alawi, R.; Stonham, T. (1989) A training strategy and functionality Analysis of Multilayer Boolean Neural Networks. Dep. of Electrical Eng. and Electronics. Brunel University.

[Arbib, 87] Arbib, M.A (1987) Brains Machines and Mathematics. McGraw-Hill Book Company, New York.

[Bledsoe-Browning, 59] Bledsoe, W.; Browning, I. (1959) Pattern recognition and reading by machine. *Proc. Eastern Joint Computer Conference*, Boston, pp. 225-232.

[Block, 62] Block, H. (1962) The perceptron: A model for brain functioning I. *Reviews of Modern Physics*, **34**, pp.123-135. [Booth, 71] Booth, T.L. (1971) Digital Networks and Computer Systems. John Wiley and Sons, INC.

[Bruce-Fu, 63] Bruce, G.D. & Fu, K.S. (1963) A model for finite state probabilistic systems. *Proc. Conf. Circuit and System Theory*, Allerton,

University of Michigan Press, Ann Arbor, Michigan.

[Bush-Mosteller, 65] Bush, R.R. & Mosteller, F. (1965) Stochastic Models for Learning. Wiley, New York.

[Cheung, 73] Cheung, C.Y. (1973) Some Aspects of adaptive Logic for Pattern Recognition. PhD Thesis. University of Kent at Canterbury.

[Clarkson et. al, 89] Clarkson, T.; Gorse, D. & Taylor, J. (1989) Hardware realisable models of neural processing. *Proc. 1st IEE International Conference on Artificial Neural Networks*, London, pp.242-246.

[Dawson, 76] Dawson, C. (1976) Aspects of Simple Scene Analysis with Learning Nets. PhD Thesis. University of Kent at Canterbury.

[de Leeuw et. al, 56] (1956) de Leeuw, K; Moore, E.F.; Shannon, C.E. & Shapiro, N. Computability by probabilistic machines. In, *Automata Studies* (eds. Shannon, C.E. & McCarthy, J.). Princeton University Press, pp. 183-212.

[Fairhurst-Maia, 83] Fairhurst, M.C. & Mattoso-Maia, M.A.G. (1983) A two-layer memory network architecture for a pattern classifier. *Pattern recognition Letters* **1**, pp.267-271.

[Fairhurst, 73] Fairhust, M.C. (1973) The Dynamics of Learning in Some Digital Networks. PhD Thesis. University of Kent at Canterbury.

[Fernandes, 77] Fernandes, C.G. (1977) Adaptive Sequence Recognition with Memory Elements. PhD Thesis. University of Brunel.

[Fernandes, 85] Fernandes, C.G. (1985) Stability Properties Inherent to Digital Neural Networks. *Proceedings of COGNITIVA 85*, Paris.

[Freeman, 61] Freeman, H. (1961) On the Encoding of Arbitrary Geometric Configurations. *IEE Trans. Elect. Comput*, **EC-10**, pp.260-268.

[Fu-Li, 69] Fu, K.S. & Li, T.J. (1969) On stochastic automata and languages. *Information Sciences* **1**, pp. 403-419.

[Fu, 82] Fu, K.S. (1982) Syntactic pattern recognition and applications. Prentice-Hall, Inc., Enlewood Cliffs, N.J.

[Fukushima, 73] Fukushima, K. (1973) A model of Associative Memory in the Brain. *Kybernetik* **12**, pp. 58-63.

[Giles et al., 90] Giles, C.L.; Sun, G.Z.; Chen, H.H.; Lee, Y.C. & Chen, D. (1990) Higher order recurrent networks & grammatical inference, to appear in *Advances in Neural Information Processing Systems 2* (ed. D.S. Toureyzky). Morgan Kaufmann.

[Gorse-Taylor, 88] Gorse, D. & Taylor, J. (1988) On the equivalence and properties of noisy neural and probabilistic RAM nets. *Physics Letters A*, **131**(6), pp.326-332.

[Grossberg, 76] Grossberg, S. (1976) Adaptive pattern recognition and universal recording: Part I, parallel development and coding of neural feature detectors. *Biol. Cybernetics* **23**, pp.121-134.

[Guiasu, 68] Guiasu, S. (1968) On Codification in finite abstract random automata. *Information and Control* **12**, pp. 227-283.

[Hebb, 49] Hebb, D. (1949) *The Organization of Behavior*. Chapman and Hall, London. [Hinton et. al, 84] Hinton, G.E.; Sejnowski, T.J. & Ackley, D.H. (1984) Boltzmann Machines: Constraint Satisfaction Networks that learn. Technical Report CMU-CS-84-119, Department of Computer Science, Carnegie-Mellon University.

[Hopcroft-Ullman, 79] Hopcroft, J.E. & Ullman, J.D. (1979) Formal Languages and their relation to Automata. Addison-Wesley Publishing.

[Hopfield, 82] Hopfield, J. (1982) Neural networks and physical systems with

emergent collective computational abilities. *Proc. National Academy of Science USA*, **79**(8), pp. 2554-2558.

[Judd, 90] Judd, J.S. (1990) Neural Networks Design and Complexity of Learning. MIT Press.

[Kan-Aleksander, 89] Kan, W.K. & Aleksander, I. (1989) RAM-Neurons for Adaptive Image Transformation Tasks. In *Neural Computing Architectures* (ed. Aleksander, I.). Chapman and Hall, London.

[Kauffman, 69] Kauffman, S.A. (1969) Metabolic Stability and Epigenisis in Randomly Constructed Genetic Nets. *Journal Theorectic Biology*, **22**, pp.437,467.

[Kleene, 56] Kleene, S.C. (1956) Representation of events in nerve nets and finite automata. In, *Automata Studies* (eds. Shannon, C.E. & McCarthy, J.). Princeton University Press, pp.3-41.

[Knast, 69a] Knast, R. (1969) Linear probabilistic sequential machine. *Information and Control* **15**, pp.111-129.

[Knast, 69b] Knast, R. (1969) Continuous-time probabilistic automata. *Information and Control* **15**, pp.335-352.

[Kohonen, 89] Kohonen, T. (1989) Self-Organization and Associative Memory. Springer-Verlag.

[Lucas-Damper, 89] Lucas, S.M. & Damper, R.I. (1989) A New Learning Paradigm for Neural Networks. *Proc. 1st IEE International Conference on Artificial Neural Networks*, London, pp. 346-350.

[Ludermir, 86] Ludermir, T.B. (1986) Sequence Discrimination with Digital Neural Networks (in Portuguese). M.Sc. Thesis. Departamento de Informatica, UFPE, Brazil.

[Ludermir, 88] Ludermir, T.B. (1988) Pattern Recognition using Neural Network. Imperial College. Neural Computing Group Internal Report.

[Ludermir, 89a] Ludermir, T.B. (1989) A Feedback RAM-Network for Temporal Pattern Recognition. Neural Systems Enginnering Internal Report.

[Ludermir, 89b] Ludermir, T.B. (1989) Stability and Temporal Pattern Recognition with Feedback RAM-Network. Neural Systems Engineering Internal Report.

[Ludermir, 90a] Ludermir, T.B. (1990) Stability and Temporal Pattern Recognition. *Proceedings IJCNN-90*, pp. 428-431. Washington.

[Ludermir, 90b] Ludermir, T.B. (1990) A Feedback Network for Temporal Pattern Recognition. In, *Parallel Processing in Neural Systems and Computers* (eds. R. Eckmiller, G. Hartmann, G. Hauske). North-Holland, Amsterdam, pp. 395-398.

[Ludermir, 90c] Ludermir, T.B. (1990) Computability of Boolean Networks. Neural Systems Engineering Internal Report.

[McClelland-Elman, 86] McClelland, J.; Elman, J. (1986) Interactive processes in speech processing: The TRACE model. In, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (eds. J. McClelland, D. Rumelhart). MIT Press, London, vol. 2, pp. 58-121.

[McCulloch-Pits, 43] McCulloch, W.; Pitts, W. (1943) A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, **5**, pp. 115-133.

[Minsky-Papert, 69] Minsky, M.; Papert, S. (1969) *Perceptrons: An Introduction to Computational Geometry*. MIT Press, London (2nd edition, 1989).

[Mozer, 88] Mozer, M.C. (1988) A Focused Back-Propagation Algorithm for

Temporal Pattern Recognition, Technical report CGR-TR-88-3, Dep. of Psychology and Computer Science, University of Toronto.

[Myers, 88] Myers, C. (1988) Learning Algorithms for Probabilistic Neural Nets. *Proceedings First INNS Annual Meeting*, Boston

[Nappey, 77] Nappey, J.A. (1977) Aspects of N-tuple Character Recognition for a Blind Reading Aid. PhD Thesis. University of Kent at Canterbury.

[Page, 69] Page, C.V. (1969) Strong stability problems for probabilistic sequential machines. *Information and Control* **15**, pp.487-509.

[Paz, 66] Paz, A. (1966) Some aspects of probabilistic automata. *Information and Control* **9**, pp.26-60.

[Paz, 71] Paz, A. (1971) Introduction to Probabilistic Automata. Academic Press, 1971.

[Porat-Feldman, 88] Porat, S. & Feldman, J.A. (1988) Learning automata from ordered examples. Technical report TR 241, Department of Computer Science, University of Rochester.

[Rabin, 63] Rabin, M.O (1963) Probabilistic automata. *Information and Control* **6**, pp.230-245.

[Reeves, 74] Reeves, A.P. (1974) A Digital Learning System for Tracking Pattern Features. PhD Thesis, University of Kent at Canterbury.

[Rumelhart et. al, 86] Rumelhart, D.; Hinton, G.; Williams, R. (1986) Learning internal representations by error propagation. In, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (eds. D. Rumelhart, J. McClelland). MIT Press, London, vol. 1, pp. 318-362.

[Rosenblatt, 58] Rosenblatt, F. (1958) The perceptron: A probabilistic model for information storage and organization in the brain. *Psychology*

Review, **65**, pp. 386-408.

[Salomaa, 69] Salomaa, A. (1969) Probabilistic and weighted grammars. *Information and Control* **15**, pp.529-44.

[Shannon-Weaver, 49] Shannon, C.E. & Weaver,W. (1949) The Mathematical Theory of Communication. University of Illinois Press.

[Shannon-McCarthy, 56] Shannon, C.E. & McCarthy, J. (1956) Automata Studies. Princeton University Press.

[Stonham, 74] Stonham, T.J. (1974) The Classification of Mass Spectra with Adaptive Logic Networks. PhD Thesis. University of Kent at Canterbury.

[Stornetta et. al, 87] Stornetta, W.S.; Hogg, T. & Huberman, B.A. (1987) A Dynamical Approach to Temporal Pattern Processing. *Proc. of the IEEE Conference on Neural Information Processing Systems*, pp. 750-759.

[Tattersall, 89] Tattersall, G.; Foster, S.; Linford, P. (1989) Single-layer look-up perceptrons. *Proc. 1st IEE International Conference on Artificial Neural Networks*, London, pp. 148-152.

[Taylor, 87] Taylor, J.G. (1987) Noisy neural net states and their time evolution. King's College report, London.

[Tollyfield, 75] Tollyfield, A.J. (1975) Aspects of Training and Connection in Some Cellular Learning Network. PhD Thesis. University of Kent at Canterbury.

[Tsetslin, 61] Tsetslin, M.L. (1961) On the behaviour of finite automata in random media. *Automat. Remote Control* **22**, pp. 1345-1354.

[Turakainen, 68] Turakainen, P. (1968) On stochastic languages. *Information and Control* **12**, pp.304-13.

[Turing, 36] Turing, A.M. (1936) On computable numbers with an application to the Entscheidungs problem. *Proc. London Mathematical Society, Ser. 2,* **42**, pp. 230.

[Vidal, 88] Vidal, J. (1988) Implementing neural nets with programmable logic. *IEEE Trans. on Acoustics, Speech and Signal Processing*, **36**(7), pp. 1180-1190.

[von Neumann, 51] von Neumann, J. (1951) The general and logical theory of automata. In *Cerebral Mechanisms of Behavior : The Hixon Symposium.* pp.1-31. John Wiley & Sons.

[von Neumann, 56] von Neumann, J. (1956) Probabilistic logic and the synthesis of reliable organisms from unreliable components. In, *Automata Studies* (eds. Shannon, C.E. & McCarthy, J.). Princeton University Press, pp. 43-98.

[Widrow, 62] Widrow,B. (1962) Generalization and information storage in networks of ADALINE neurons. In *Self-Organizing Systems* (ed. Yovits, G.T.). Spartan Books, New York.

APPENDIX 1

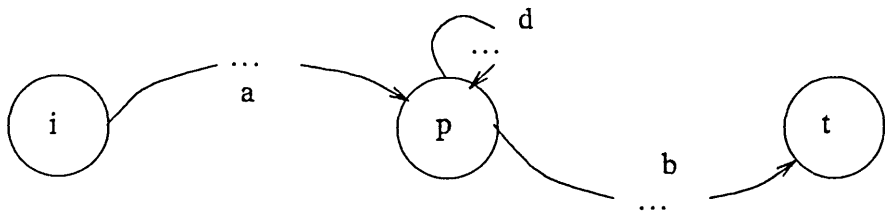
LANGUAGE L IS NOT REGULAR

Here we are going to prove with all details that the language $L=\{(1^m01^n) \mid m \leq n\}$ is not regular. This example is used throughout this thesis every time we need a weighted regular language which is not regular.

In order to prove that L is not regular we need first to introduce some definitions and to prove some others facts. In what follows we will give all we need.

Proposition 1 - If L is a regular language, there exists an integer n such that if $s \in L$ and $|s| \geq n$, then s admits a factorisation uwv with $|w| \neq 1$ such that $uw^*v \subset L$. Furthermore, w can be chosen as a subsegment of any segment of s of length n. In particular $|w| \leq n$.

Proof - Let A be an automaton with n states which recognises L. Let c be a word in L and c' be a subword of c of length exactly n. Since the successful path of c' in A must contain a repeated state, then results a factorisation of c



with d a subword of c' and $|d| \neq 1$. Setting $u=a$, $v=b$ and $w=d$, it follows that $uw^*v \subset L$.

If L is an infinite language, L contains a word s with $|s| \geq n$ and the successful path for s in A will then involve a repeated state.

Corollary 2 - If L is an infinite regular language, then $uw^*v \subset L$ for some

$u,w,v \in \Sigma^*$, with $|w| \neq 1$.

This corollary permits us to show that various languages are not accepted by finite state automaton.

Definition 3 - Given an automaton $A=(\Sigma,Q,\delta,q_0,F)$, the reversal automaton is $A^\phi=(\Sigma,Q,\delta',q_0,F)$ with $\delta'(p,a)=q$ for each $\delta(q,a)=p$ in A. Thus a word w in A with $w=w_1,\ldots,w_n$ yields a word w^ϕ in A^ϕ with $w^\phi=w_n,\ldots,w_1$.

Proposition 4 - The class of regular languages is closed under union.

Proof - Let L_1 and L_2 be regular languages generated by regular grammars $G_1=(V_N^1,V_T^1,P_1,S_1)$ and $G_2=(V_N^2,V_T^2,P_2,S_2)$, respectively. By renaming symbols, if necessary, we can assume that V_N^1 and V_N^2 contain no symbol in common, and that S is in neither of them. We construct a new grammar, $G_3=(V_N^1 \cup V_N^2 \cup S,\ V_T^1 \cup V_T^2,P_3,S)$, where P_3 consists of the productions of P_1 and P_2 except for $S_1 \to \epsilon$ or $S_2 \to \epsilon$, plus all productions of the form $S \to \alpha$ such that either $S_1 \to \alpha$ is in P_1 or $S_2 \to \alpha$ is in P_2. $L(G_3)=L(G_1) \cup L(G_2)$.

Proposition 5 - The class of sets accepted by finite state automaton is closed under complement.

Proof - Let $M_1=(\Sigma_1,K,\delta,q_0,F)$ be an automaton accepting a set S_1. Let Σ_2 be a finite alphabet containing Σ_1 and let d be a new state not in K. We construct M_2 to accept $\Sigma_2^* - S_1$. Let $M_2=(K \cup d,\Sigma_2,\delta_2,q_0,(K-F) \cup d)$, where $\delta_2(q,a)=\delta_1(q,a)$ for each q in K and a in Σ_1, $\delta_2(q,a)=d$ for each q in K and a in $\Sigma_2 - \Sigma_1$, and $\delta_2(d,a)=d$ for each a in Σ_2. M_2 accepts $\Sigma_2^* - S_1$.

Proposition 6 - The class of regular languages is closed under intersection.

Proof - Immediately from the fact $L_1 \cap L_2=\overline{\bar{L}_1 \cup \bar{L}_2}$ and propositions 4 and 5.

Proposition 7 - The set $L = \{(1^p 01^p) \mid p \geq 0\}$ is not regular.

Proof - If L is regular, it would have to contain uw^*v for some $u,w,v \in \Sigma^*$, with $|w| \neq 1$. Clearly w cannot contain 0 as a letter since all words of L have only a single 0. Thus $w = 1^n$ for some $n > 0$. Either u or v (but not both) must contain 0. Thus assume $u = 1^p$, $v = 1^r 01^s$. Then $uw^*v = \{1^{p+kn+r}01^s \mid k \geq 0\}$. To be in L we must have $p + kn + r = s$ for all $k \geq 0$. This is impossible since $n \neq 0$. Similarly the possibility $u = 1^r 01^s$, $v = 1^p$ is excluded.

Proposition 8 - The language $L = \{(1^m 01^n) \mid m \leq n\}$ is not regular.

Proof - If L is regular, then so is the language $L_1 = \{(1^m 01^n) \mid n \leq m\}$ by reversal and interchanging the roles of 1 and 0. Therefore $L \cap L_1$ is regular. However, $L \cap L_1 = \{(1^m 01^n) | m \leq n\} \cap \{(1^m 01^n) | n \leq m\} = \{(1^p 01^p) | p \geq 0\}$. $\{(1^p 01^p) | p \geq 0\}$ is not regular so $\{(1^m 01^n) | m \leq n\}$ is not regular.

APPENDIX 2

PUBLISHED PAPERS

Part of the results of this thesis have been written into papers and submitted to international conferences on neural networks. The first paper [Ludermir, 90a], with the title "Stability and Temporal Pattern Recognition", was published in the proceedings of the International Joint Conference on Neural Networks which was held in Washington in January 1990. This paper contains the results of section 4.5 of this thesis. The second paper [Ludermir, 90b] with the title "A Feedback Network for Temporal Pattern Recognition", was published in the book *Parallel Processing in Neural Systems and Computers* edited by R. Eckmiller, G. Hartmann and G. Hauske. This paper contains most of the results in chapter 4 of this thesis. These two papers are attached with this appendix.

Stability and Temporal Pattern Recognition

*TERESA B. LUDERMIR**
Neural Systems Engineering Group
Imperial College, London SW7 2BT, England
email: JANET tbl%winge@sig.ee.ic.ac.uk

ABSTRACT

The aim of this paper is to discuss the influence of the stability property in the generalization of a neural net and consequently in the performance of the net to solve a specific task. The task we are working with is the recognition of temporal patterns and the model employed is an artificial neural net based on RAM as digital neurons[Aleksander, 79]. Some ways to control the stability of the net are presented. Experiments were done with different methods of controlling the stability and some of them are presented here.

1. INTRODUCTION

There are different types of neural nets. The study of neural nets were largely originated in 1943 with the McCulloch and Pitts model of neuron [McCulloch-Pitts, 43]. They proposed a neuron model implemented by threshold logic gates, where variable input weights play a role analogous to that of synapses in natural neurons. The model used here is based on a different model called the RAM neuron model. The RAM model is based on the simple operations of a look-up table which is best implemented by random access memory (RAM) and where the knowledge is directly "stored" in the memory (the look-up tables) of the nodes during learning. Some advantages of this model are: (1) it is straightforward to implemented in hardware; (2) learning is not unreasonably slow and (3) error-correction requires only a global success signal.

The most important property of a pattern recognizer is generalization. Generalization is the ability to classify patterns others than those in the training set. RAM-nets having feedback connections between neurons have been successful with some temporal pattern recognition tasks [Ludermir, 89] but feedback machines are more sensitive to input errors than feedforward machines. However RAM-feedback nets are inherently stable adaptive structures [Fernandes, 85]. They are able to recover from input errors naturally and capable of recognizing input sequences independent of its initial state.

2. STABILITY AND GENERALIZATION PROPERTIES OF NETS WITH FEEDBACK

The type of net used in this work consists of a layer of identical RAM type digital neurons, where each of them has n-address terminals, i connected to an external matrix of binary elements and f connected to the output terminal of others neurons through clocked delay units ($n = i+f$). The RAM type digital neuron is represented in the figure 2.1.

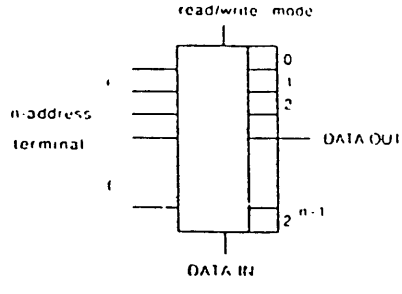


Figure 2.1 RAM type digital neuron

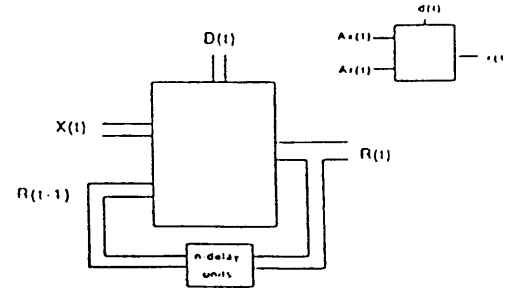


Figure 2.2 Sequential Digital Neural Net

The structure of a SDNN (Sequence Digital Neural Network) is represented in the figure

Supported by CNPq (Brazilian Research Council) grant no. 20.3296/86-CC

2.2 where binary vectors x(t), r(t), r(t-1) and d(t) represent respectively at time t, the state of input matrix, the response, the delayed response and the 'desired' response of the neurons. The input and feedback connections are randomly generated. The SDNN is trained to anticipate its inputs, i.e. $d(t)=x(t+\alpha)$, such that $r(t)=x(t+\alpha)$ during test phase. During the training phase the net is fed with $\bar{x}\in A$ (where A is the training set) with one (or more) RAM(s) in the write mode and the memory position is changed $M_i[A_x(t),A_r(t)]=d_i(t)$ (A_x and A_r are input and feedback component respectively) for all RAMs in the write mode.

The stability property of the net is responsible for the increase of generalization. Thus it has direct influence on pattern discrimination and identification. Three types of misclassification can come from the generalization of the net. 1) rejection by doubt (intersection of two or more generalization sets); 2) unknown rejection (a pattern $\bar{x} \in L$ fall outside the generalization set); 3) error (a pattern fall within the generalization set of another category).

The stability of RAM-nets mainly depend on two parameters: a) neuron memory contents and b) feedback connection.

a) The stability can be controlled by distribution of 0's and 1's in the neuron memory. The greatest the difference between the number of zeros and ones in the neuron memory less will be the possibility of changes in the neuron output, in consequence the net will be more stable. There are two different ways of controlling the memory contents. 1) direct control: The distribution of zeros and ones is made randomly based on the difference L between zeros and ones and 2) adaptive control: The distribution of zeros and ones is made through training with any training strategy. These were first used by Fernandes in [Fernandes, 85].

b) Feedback connection influences the recover of a input error through time in a net. If we have a small feedback connection the error propagation through time is going to be reduced and the recover phase will be small. Thus the net is more stable.

3. INFLUENCE OF STABILITY IN GENERALIZATION AND IN PATTERN RECOGNITION

A classifier based on the probability $p(\bar{x}\in L_k/\bar{r})$ that the input sequence $\bar{x}\in L_k$ given that the response $\bar{r}$ occurred when $\bar{x}$ was fed into the net $0<=p(\bar{x}\in L_k/\bar{r})<=1$ was used. Although the ideal would be $p(\bar{x}\in L_k/\bar{r})=1$ for all symbols in the sequence $\bar{x}\in L_k$, this does not generally happen. What happens is that $p(\bar{x}\in L_k/\bar{r})$ changes randomly near 1 for sequences $\bar{x}$ in L_k and near 0 for sequences $\bar{x}$ not in L_k. In consequence, a measure is required which considers $p(\bar{x}\in L_k/\bar{r})$ for all input symbols in the same sequence $\bar{x}$. The measure that is being used is the continuous average

$$S_k(t)=a^{-1}\sum_{t=1}^{a} p(\,x\in L_k/r(t)),k=1,2 \text{and} 0\leq S_k<=1.$$

The measures we used to analyse the results are the size of state sets in each class of sequences fed into the net, the percentage of error recovery of the input sequence and sequence distinction. We do not wish neither that states sets of each class being large (for saving process time and memory) nor that the common states being in large quantity (for better distinction). The percentage of error recovery of input sequence has its importance in the states which occurs with a input sequence. Once the hamming distance between the responses of the sequence $\bar{x} \in L$ fed into the net in training and the responses of the sequence $\bar{x}' \in L$ fed into the net in testing is zero ($h(t)=h[r(t),r'(t)]=0$) all the responses will be the same.

Experiments were done to distinguish different geometric forms, such as triangles, squares and circles. The nets used were MNDS(32,2,2), MNDS(32,2,3), MNDS(32,2,4), MNDS(32,2,5) and MNDS(32,2,6). The distribution of 0's and 1's in the neuron memory were controlled by direct control and adaptive control as described in the last section. The input patterns $\bar{x}$ correspond to a sequence of eight tracking movements.

In the figure 3.1 below we show the percentage of error recovery of input sequences for experiments with five different nets. As we can see in the figure when the feedback connection increases the percentage of error recovery of input sequences decrease. With

MNDS(32,2,2) there is no difference between a direct control of stability with high value of L and adaptive control because the feedback connection is small and as consequence the net is very stable. With MNDS(32,2,3) the net with direct control with L=0, that is half of the memory contents is zero and half is one, is not able to recover from input error at all because the net was made unstable from the very small difference between the number of zeros and ones and also from the increase of the feedback connection. From MNDS(32,2,4) as the net is not very stable because of the high feedback connection only the adaptive control is successful in making the net to be able to recover from input errors.

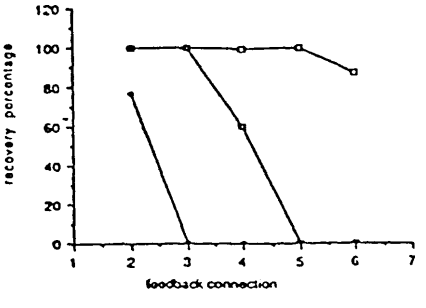


Figure 3.1 Recovery Percentage

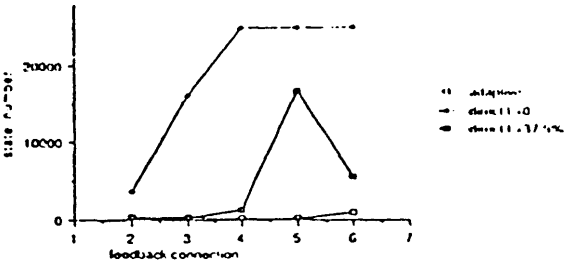


Figure 3.2 State Sets Size

It is important to observe that nets with high differences between the number of zeros and ones in memory of RAMs proceeding from direct control of stability not always result in a high input error recovery of the net. The reason for this is because this increase in the differences between zeros and ones were done randomly.

In the figure 3.2 we show the state quantity in each class of sequences for experiments with five different nets. The maximum number of states by class is the number of sequences fed into net times the number of symbols in each sequence: $500 \times 50 = 25.000$. With MNDS(32,2,2) when using adaptive control we have smaller states set, that is less computation is necessary to discriminate sequences in different classes, than when using direct control. With MNDS(32,2,3) the size of the state sets are closer with adaptive control and direct control with high value of L. From MNDS(32,2,4) we have very big state sets with direct control while with adaptive control we still have a reasonable size. The size of states set will have effect upon sequence discrimination as we will see in the next paragraph. Experiments in which the size of the state sets are big we have no discrimination among the sequences belonging to different classes.

Below we show in the figure 3.3 the discrimination capability of the net for experiments with five different nets. The discrimination capability of the net is illustrated by the difference between the values of $S_1(t)$ from sequences in the class we want to recognize and the values of $S_1(t)$ from sequences not in the class we want to recognize. When such values are negative it means that there was an intersection between the values and the number means the size of the intersection. With MNDS(32,2,2) and MNDS(32,2,3) were possible to distinguish the sequences with all ways of controlling the stability of the net because the nets are stable in all cases and the state sets are not very big.

It should be noticed that with MNDS(32,2,2) there was not to much difference in the input error recover between the two methods of stability control and the size of the states sets with adaptive control was smaller than with direct control meaning that the adaptive control is much efficient in sequence discrimination. However with MNDS(32,2,3) the size of the state sets are closer with adaptive control and direct control with high value of L, the input error recovery was similar and the discrimination power between the two ways of controlling the stability of the nets are also the same. From MNDS(32,2,4) we have values of $S_1(t)$ very close to each other mainly in the cases where the stability control was direct with L=0. With high value of feedback connection the net is not very good in discriminating sequences.

When we increase the feedback connection the net will remember of a input error for a longer time and we have a less stable structure. The percentage of input error recovery will

deteriorate. We are going to have big state sets and consequently in some cases we are going to have difficult in temporal pattern recognition.

With unstable net the recognition of pattern is difficult also because small variations between the prototypes (patterns in training set) and pattern in the test set will result in a completely different response sequence r. An unstable net will generate more unknown rejection whilst a very stable one will generate more rejection by doubt (net is not able to notice the differences between patterns) and error with the generalization.

With more stable nets we have more number of patterns not in the training set being recognized which implies in a bigger generalization. But we need to have limit in the size of the generalization set. Big generalization sets generates more mistakes with the generalization.

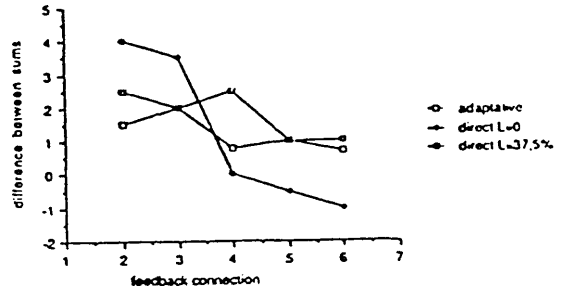


Figure 3.3 Sequence Discrimination

4. CONCLUSION

A study of the relation among the inherent stability property, the generalization and temporal pattern recognition was made. The main ideas behind this methodology are: (1) the use of a feedback RAM- net; (2) probabilistic classifier to temporal pattern recognition; (3) different methods of stability control and (4) the use of different parameters to analyse the results. Three parameters (size of state sets and their intersection, the percentage of error recovery of input sequence and sequence distinction) have been presented.

The main strengths of the method are that: (1) a response of a RAM-net with feedback carries information about the order of appearance of its input patterns; (2) the RAM-net is capable of recognizing patterns independently of its initial states even in the presence of input distortions and (3) the generalization emergent from these nets.

Many aspects of this methodology remain to be investigated since alternative training strategy were not explored. In addition probabilistic logic neuron [Aleksander, 88] could be adopted instead of RAM. The probabilistic logic neuron can avoid knowledge being overwritten in the training phase and introduces some non determinism into the system.

REFERENCES

[Aleksander, 79] Aleksander, I. & Stonham, T.J.: "A Guide to Pattern Recognition using Random Access Memories".IEE J Comp & Digital Tech 2(1), 29-40, 1979.

[Aleksander, 88] Aleksander, I.: "The logic of connectionist system" in R. Eckmiller, Chr. v.d. Malsburg, eds. Neural Computers. Berlin: Springer-Verlag, 189-197, 1988.

[Fernandes, 85] Fernandes, C.G.:"Stability Properties Inherent to Digital Neural Networks", COGNITIVA 85, Paris, 1985.

[Ludermir, 89] Ludermir, T.B.:"A Feedback RAM-Network for Temporal Pattern Recognition", Neural Systems Eng Report, Imperial College, Dept of Elec Eng 1989.

[McCulloch-Pitts, 43] McCulloch, W.S. & Pitts, W.:"A logical calculus of the ideas imminent in neural nets". Bull. Math. Biophys. 5, 1943.

Parallel Processing in Neural Systems and Computers
R. Eckmiller, G. Hartmann and G. Hauske (Editors)

A FEEDBACK NETWORK FOR TEMPORAL PATTERN RECOGNITION

Teresa B. LUDERMIR*

Neural Systems Engineering Group
Imperial College, London SW7 2BT, England
email: JANET tbl%winge@sig.ee.ic.ac.uk

The aim of this paper is to show that the response of a neural network with feedback carries information about the order of appearance of its input patterns. They are capable of recognising some temporal patterns independently of their initial states even in the presence of input distortions. The model employed is an artificial neural network based on RAM memory devices as digital neurons (Aleksander [1]). The architecture has been used successfully in many sequence recognition problems. Some of them are presented here.

1. INTRODUCTION

Time is the essence of many pattern recognition tasks, e.g. speech recognition, motion detection and signature verification. However, connectionist learning algorithms to date are not well-suited for dealing with time-varying input patterns. In this paper it is shown that RAM networks are able to store sequential information from their input patterns and they operate efficiently on some temporal pattern recognition tasks.
Some weighted-connectionist systems have been implemented to deal with temporal recognition. Most of them use a buffer to hold the n most recent elements of the input sequences. Some drawbacks of this approach are: 1) the buffer must be sufficient in size to accommodate the longest possible input sequence; 2) by making a great deal of information simultaneously available, much computation is required each time; 3) each element of the buffer must be connected to a higher layer of the network. In consequence a large number of training examples must be used. Others systems are based on the fact that, in a connectionist network, the connections from one set of units to another implement a mapping. This approach is rather inflexible in that the mapping function used to construct the temporal context is predetermined and fixed. As a result, the representation of a temporal context must be made sufficiently rich to accommodate a wide variety of tasks.
The model used here is based on a different model called the RAM neuron model. The RAM model is based on the simple operations of a look-up table which is best implemented by random access memory (RAM) and where the knowledge is directly "stored" in the memory (the look-up tables) of the nodes during learning. Some advantages of this model are: (1) it is straightforward to implement in hardware; (2) learning is not unreasonably slow and (3) error-correction requires only a global success signal. An example of such a model is the pattern recognition device called WISARD as in Aleksander [2].

2. SEQUENTIAL BEHAVIOUR OF RAM NETWORKS WITH FEEDBACK

The type of network used in this work consists of a layer of identical RAM type digital neurons, where each of them has n-address terminals, i of them connected to an external matrix of binary elements and f of them connected to the output terminal of others neurons through clocked delay units ($n=i+f$). The RAM type digital neuron is represented in figure 2.1. The structure of a SDNN (Sequence Digital Neural Network) is represented in figure 2.2 where binary vectors x(t), r(t), r(t-1) and d(t) represent respectively at time t, the state of input matrix, the response, the delayed response and the 'desired' response of the neurons. The input and feedback connections are randomly generated. The SDNN is trained to anticipate its inputs, i.e. $d(t)=x(t+\alpha)$, such that $r(t)=x(t+\alpha)$ during test phase. During the training phase the net is fed with $x \in A$ (where A is the training set) with one (or more) RAM(s) in the write mode, and the memory position is changed $M_i[A_x(t),A_r(t)]=d_i(t)$ (A_x and A_r are input and feedback component respectively) for all RAMs in the write mode.

Supported by CNPq (Brazilian Research Council) grant no. 20.3296/86-CC

In order to justify the use of a single layer network(SLN) in sequence discrimination it will be shown that the response of SLN carries information about the order of appearance of its input patterns; that is, each output of the network depends not only on the present input but also on the previous ones.

Theorem : r(t) is function of x(1) x(2) ... x(t).
Proof : Let x = x(1) x(2) ... x(a) be the input sequence and r = r(1) r(2) ... r(a) be the corresponding response sequence of the network. Obviously, in general, x(i)'s do not relate themselves, since each sequence x is simply an element of the free monoid generated by the input symbols. Then, it is necessary to show only that r(t), $1<t<a$ is function of x(1) x(2) ... x(t) which gives to each r(t) of r the status of sequentiality that we are looking for. There follows a proof by induction on the length of r. r(1) = (x(1), r(0)) by definition. Suppose now that r(t) = (x(t) ... x(1), r(t-1)), then by definition r(t+1) = (x(t+1), r(t)), which gives us r(t+1) = (x(t+1) x(t) ... x(1), r(t)). Thus r(t) is function of all inputs until the instant t, for any t, as stated. Observe also that r(t) is function of r(t-1), which characterises networks with feedback.
It was shown above that the response of a net with feedback carries information about the order of appearance of its input patterns. But feedback nets are not the only way to deal with this problem. It is possible to use a feedforward RAM-net with more inputs at each time. The advantage of feedforward RAM-nets is that they are less sensitive to input errors, although it would be necessary to hold and process more information each time.
From the study of the stability properties of feedback networks in Fernandes [4] it is known that such networks are able to recover from input errors naturally. They are able to recover from input errors independently of their initial states. It is also known that the stability of the network is responsible for the increase of generalisation. Thus it has direct influence on pattern discrimination and identification. It is possible to control the stability of the network in at least two different ways: by altering memory contents and by the influence of the feedback connection as in Ludermir [6].

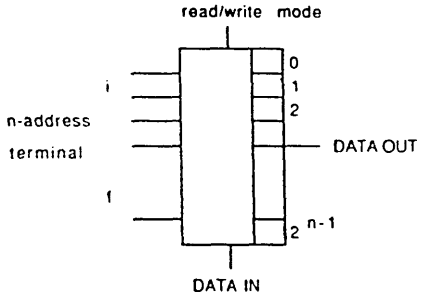


Figure 2.1 RAM type digital neuron

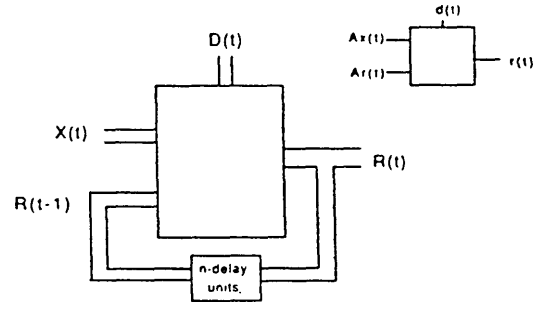


Figure 2.2 Sequential Digital Neural Net

3. EVALUATION OF THE FEEDBACK RAM-NETWORK

In this section three different types of classifiers are presented all based on the RAM-network described in the last section. The use of different classifiers gives a better insight into the system and shows the importance of the RAM-network itself. Also the choice of the classifier depends on the desirable application one has in mind. With all classifiers we did two kinds of experiments. The first one was to distinguish between different geometric forms, such as triangles, squares and circles. The second one was to recognize regular languages. Due to lack of space, only the results of the second set of experiments one presented here. The others results are presented in Ludermir [5]. The network used in the examples in this paper was composed of 32 RAMs with four inputs each, two from the input patterns and two from the feedback state. And the languages used were $L_1=\{x \mid x=a^i b^j c^k d^l, i,j,k,l \geq 0\}$ and L_2 is the complement: $L_2=U-L_1$, where U is the Universal set.

PROBABILISTIC CLASSIFIER

A classifier based on the probability $p(x \in L_k/r)$ that the input sequence $x \in L_k$ given that the response r occurred when x was fed into the net was used. Although ideally $p(x \in L_k/r)=1$ for all symbols in the sequence $x \in L_k$, this does not generally happen. What happens is that $p(x \in L_k/r)$ changes randomly near 1 for sequences x in L_k and near 0 for sequences x not in L_k. In consequence, a measure is required which considers $p(x \in L_k/r)$ for all input symbols in the same sequence x. The measure that is being used is the continuous average

$$S_k(t) = a^{-1} \sum_{t=1}^{a} p(x \in L_k / r(t))$$

where k=1,2 are the different languages. This measure ensures $0 \leq S_k \leq 1$. Then, for sequences belonging to the language L_k the sum $S_k(t)$ should be close to one, while for sequences not belonging to the language L_k the sum $S_k(t)$ should be close to zero.
In the example in figure 3.1 we show the results obtained with the probabilistic classifier using the languages L_1 and L_2 as defined above. This classifier is effective for distinguishing different geometric forms (Ludermir[5]). In the examples with regular languages, however, the discrimination is not so effective. One of the reasons for this is that more sequential information needs to be processed. As a result the use of a buffer classifier is proposed in such cases.

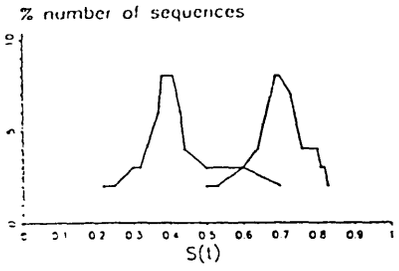


Figure 3.1 Probabilistic Classifier.

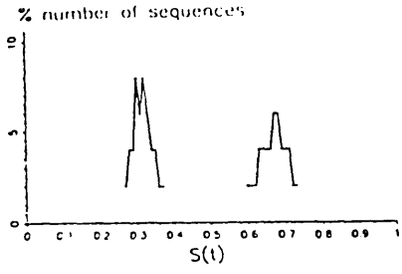


Figure 3.2 Buffer Classifier

BUFFER CLASSIFIER
The SDNN used in this work is blind to small differences in patterns. And also the feedback state does not contain enough information to recognise those small differences. In order to store more information a buffer is introduced in the probabilistic classifier. The buffer stores the time of occurrence of the states. Then the only difference between the classifiers is that here it is necessary to store the times of occurrence of the states.
It is not likely that the states in the sequences x of any class appear in the same order of the states in the training sequences for this class. The definition of a window, in which the states can occur, is needed. If the size of the window is the length of the sequence, this classifier becomes the probabilistic classifier. The sum is defined as:

$$S_k(t) = a^{-1} \sum_{t'=1}^{a} p(x \in L_k / r(t'))$$

where t'=t-t'', t''=0,1,...,ws with ws being the window size and k=1,2 ($0 \leq S_k \leq 1$ is conserved).
The results in figure 3.2 were obtained with this classifier using L_1 and L_2.
Better results were achieved with the buffer classifier than with probabilistic classifier but it usually requires the processing of much more information and the system is more complex and also takes longer to run. Despite theses drawbacks, this classifier has two properties in common with any model of temporal pattern recognition. Firstly, some memory of the input history is required. And secondly a function must be specified to combine the current memory (or the temporal context) and the current input to form a new temporal context.

FINAL STATE CLASSIFIER
In this model ideas from Formal Language Theory are used to recognise temporal patterns. It is desirable to characterise the (class of) languages(s) that can be recognised by a specific neural net. At this point only finite state languages have been used.
One way to recognise temporal patterns using neural networks is by using the final state of the network; finite automata work in the same way. Instead of probabilistic discriminations we have a set of final states.
Although the temporal transformation f:x → r examined here is a finite state representation, an analysis of the generalisation of the network is needed. The generalisation in this new classifier can be large because of the stability properties inherent in networks. After the network has been trained a finite-state structure is generated inside the network. Then the network is fed with some patterns and the final state of the network for each pattern is stored. The patterns are recognised when their final states are one of the final states stored.
The results in figure 3.3 were obtained with the final state classifier using L_1 and L_2.
As can be seen, in the examples above the results with this classifier are worse than with the previous classifiers. This is because of the large generalisation. The network recognises all sequences in the language it has been trained on but it also recognises some of the sequences

which do not belong to the language (over-generalisation). Also the network can not recognise all the finite state languages because the net is being trained only to predict the next symbol of the input sequence and there are finite state languages that do more than predict the next symbol(state). The choice of training strategy depends on the desirable application. Unfortunately, there is not a developed theory in which, for a certain kind of problem, a good training strategy can be determined.

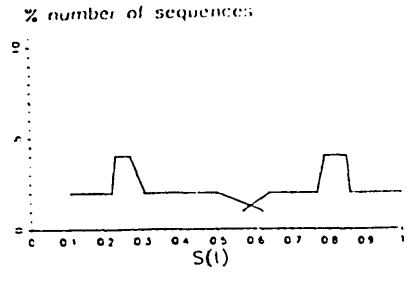


Figure 3.3 Final State Classifier.

Other motivations for this new classifier are: 1) it is very easy to know the total generalisation of the network by regular expressions; and 2) it is one way to solve problems such as parity with a single layer network with feedback.

The way that the output information of a net is processed makes a difference when working with different pattern recognition problems. Different classifiers are necessary to deal with different kinds of problems. And in order to characterise the generalisation of RAM-network, the analysis of the influence of different training strategies, classifiers, net configuration, etc are indispensable.

4. CONCLUSIONS

A methodology has been presented for Temporal Pattern Recognition. The main ideas behind this methodology are: (1) the use of a feedback RAM- network and (2) the use of different classifiers. Three different classifiers (probabilistic, buffer and final states) have been presented.

The main strengths of the method are that: (1) the response of a RAM-network with feedback carries information about the order of appearance of its input patterns and (2) the RAM-network is capable of recognising patterns independently of its initial states even in the presence of input distortions.

Many aspects of this methodology remain to be investigated since alternative training strategies were not explored. In addition probabilistic logic neurons (PLN) defined by Aleksander [3] could be adopted instead of RAM. The PLN can avoid knowledge being overwritten in the training phase and introduces some non determinism into the system.

One problem that is being analysed by the author is the the power of these feedback networks (RAM, PLN) in the recognition of languages in the Chomsky hierarchy. Different classifiers are being used in order to obtain a formal characterisation of the languages recognised by these feedback machines.

Application of this methodology to real-world problems will probably require the development of additional sequence transformations.

REFERENCES

[1] Aleksander, I. & Stonham, T.J.: "A Guide to Pattern Recognition using Random Access Memories IEE J Comp & Digital Tech 2(1), 29-40, 1979.

[2] Aleksander, I., Thomas, W.V. & Bowden, P.A.: "WISARD, a radical step forward in image recognition". Sensor Review 4(3), pp 120-124, July 1984.

[3] Aleksander, I.: "The logic of connectionist system" in R. Eckmiller, Chr. v. d. Malsburg, eds. Neural Computers. Berlin: Springer-Verlag, 189-197, 1988.

[4] Fernandes, C.G.:"Stability Properties Inherent to Digital Neural Networks", COGNITIVA 85, Paris, 1985.

[5] Ludermir, T.B.:"A Feedback RAM-Network for Temporal Pattern Recognition", Neural Systems Eng Report, Imperial College, Dept of Electrical Eng 1989.

[6] Ludermir, T.B.:"Stability and Temporal Pattern Recognition", in Proceeding IJCNN-90, Washington.