LEARNING WITH DELAYED REINFORCEMENT IN AN EXPLORATORY PROBABILISTIC LOGIC NEURAL NETWORK

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

Catherine E. Myers

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

September 1990

ABSTRACT

This thesis is concerned with neural network learning systems which:

- learn from the results of their actions, without a distinct training phase (exploratory learning)
- require only global evaluations of success (reinforcement learning)
- can cope with reinforcements which arrive only after some delay (delay learning)

First, the thesis develops a node model based on Aleksander's Probabilistic Logic (PLN) node, and which can be used for exploratory reinforcement learning. Like the PLN, it has a training algorithm which requires only a global evaluation of the success of a network output, rather than explicit provision of what the optimal action would have been. Unlike the simple PLN, it allows the training algorithm to be incremental. Therefore behaviours may be shaped gradually and on-line, and separate train and run phases are not required.

Next, Attention-Driven Buffering (ADB) is introduced as a means for performing delay learning. ADB systems can learn to predict results which occur over indefinitely long delays while maintaining a small number of past states buffered locally at the nodes. The principal idea is that states should be allocated buffer space based not only on recency but also on unpredictability. Unpredictable inputs are assigned a high "Attention", and this increases their ability to compete for buffer space. An ADB system applied to a state traversal task can learn the results of actions in a given state; it can also learn not to enter states with immediate positive reinforcement, but which lead inevitably to negative reinforcement.

Finally, an ADB system has been constructed to perform delay learning tasks resembling those to which an animal such as the octopus can be trained. The sys-

tem, OVSIM (for *Octopus Vulgaris* SIMulation), is shown not only to exhibit similar capabilities, but to share several features of its learning processes with the octopus. Further, when various functions are damaged, the changes in behaviour or learning ability are comparable to those exhibited by an octopus with removal of various brain regions. The ADB mechanism, it is argued, therefore takes on some credibility as a hypothesis of how brains accomplish delay learning. This hypothesis is strengthened by current theories that the mammalian hippocampus and amygdala perform buffering and novelty-detection, which an ADB mechanism would require.

ERRATA

Added after Examination 30.11.90

page 24: The second and third paragraphs should be written as:

There are numerous methods for training these nodes to perform desired input-output mappings. Many derive from the Hebbian formalism which is another simplification of observed bviological behaviour: "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" [Heb49, p.50]. Formally, if y_{pj} is the output of node j in some context p, and if w_{ji} is the weight to it from node i, then:

$$\Delta_p w_{ji} \propto y_{pi} y_{pj} \qquad (2.3')$$

This rule strengthens weights between nodes that are frequently coactive. It does not encourage nodes to perform some desired mapping, as it makes no mention of a desired mapping – only of coactivity between nodes.

page 33: The comparison between RAM net capacity and weighted-node net capacity is made by Wong and Sherrington in [WoS89].

page 183: The bibliography should contain the following reference:

[RGV88] Rosen, B; Goodwin, J.; Vidal, J. (1988) Learning by state recurrence detection. In, *Neural Information Processing Systems* (ed. D. Anderson). American Institute of Physics, New York, p.. 642-651.

ACKNOWLEDGEMENTS

I would like to express my deepest thanks to everyone who has given me support and advice throughout the completion of this thesis. In particular, my supervisor Prof. Igor Aleksander has been unfailingly helpful, encouraging and insightful. I am also very grateful to everyone who has been associated with the Neural Systems Engineering Group at Imperial for their friendship and discussions. Finally, thanks and love to the family and friends who all helped in so many ways, most of all by their unflagging encouragement.

This research was funded by the National Science Foundation (USA) under an NSF Graduate Fellowship.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

FREQUENTLY-USED SYMBOLS

A_{xy}	For pattern $x=(x_1 \cdots x_B)$ and $y=(y_1 \cdots y_B)$, $A_{xy}=\frac{1}{B}\sum_{i=1}^{B}(x_i=y_i)$
α	Maximum assignable attention for items in ADB buffer
α'	Maximum number of cycles an element might stay in ADB buffer, $\alpha'=\left\lceil\frac{\alpha}{\delta}\right\rceil$
$attn_i$	The current attention of the ith item in the ADB buffer, $0 \le attn_i \le 1$ (or simply *attn* for a 1-element buffer)
B	Number of bits in the external input
β	The amount by which an MPLN stored value is incremented on reinforcement; β^+ on positive reinforcement or β^- on negative
β^+, β^-	The average reinforcement to an MPLN stored value associated with a psitive (β^+) or negative (β^-) element in the ADB buffer
D	Delay between action and arrival of reinforcement
d_{pj}	Desired or target output of node j to pattern p
$\Delta_p w_{ji}$	Change to weight w_{ji} after presentation of pattern p
δ	The amount by which attention of a buffered item decays with each time step
E	Total error of the network over all training patterns
E_p	Error of the network in response to pattern p
$F(L,I)$	Number of functions computable by an L-level pyramid of I-input PLNs
f	Attention-generating function in ADB: $f(x)=[0,\alpha]$
Φ	Output function for PLN: $\Phi(i) \in \{0,1\}$

Φ^P	Output probability function for PLNs: $\Phi^P(z) \in [0,1]$
θ_j	The threshold function of node j
I	Number of inputs to a node
j	Index of a node
k	Number of bits to encode attention parameter in ADB buffers
K_{jir}	In a pRAM, the influence of node i firing r time steps ago on the current activity at node j
L	Number of levels of a network (typically a pyramid topology)
M	The set of training patterns; $M^+ \in M$ are those associated with a positive reward, $M^- \in M$ are associated with a negative reward, and $\|M\|$, $\|M^+\|$, $\|M^-\|$ are the cardinalities of these sets
n	The size of the buffer in a delay learning system
N	The number of nodes in a network
P sub i	The ith training pattern presented to the network (consists of B bits)
p_{ji}	The predicted output of node j, under input i (In Yeung's system, which constructs the desired output pattern, p_{ji} is the probability that node j under input i outputs a 1; p_{ji}* is the constructed desired p_{ji})
ψ	The number of MPLNs in the ADB learning subsystem which must output 1 in order for the system output to be "act" or "accept the input pattern"
$R_i(t)$	Response of ADB learning system to input at time t; system output generated by output-generator on the basis of this
r	Reinforcement signal, scalar to the whole network: $r \in \{-1,0,+1\}$
$\hat{r}$	In the ASE/ACE system, the internal reinforcement constructed by the ACE and used to train the ASE

$\hat{s}$	The value to which MPLN stored values are initialised before training
sv_X	A value stored in jth PLN at address X
sv_X^*	The value of sv_X at the nearest solution
sv_{ERRj}	A value at location ERR such that $sv_{ERRj} \neq sv_X^*$
u	The unknown value in a PLN, where $\Phi^P(u)=0.5$; in an MPLN, the element of Ω such that $\Phi^P(v_u) \approx 0.5$
U	An untrained pattern
v_i	The ith element of Ω, $0 \leq i \leq 1$
v_u	The element of Ω such that $\Phi^P(v_u) \approx 0.5$
V	The total number of votes (i.e., "1" outputs) from OVSIM's MCs in response to an input pattern
ω	The cardinality of Ω
Ω	The alphabet of possible stored values in an MPLN: $\Omega=(v_0 \cdots v_{\omega-1})$
x_i or x_{ji}	The ith input to node j, $0 \leq i \leq I-1$
$\bar{x}_{ji}$	Trace of the past activity of input x_{ji}
X	An address to node j, $X=(x_0 \cdots x_{I-1})$
$\|X\|$	The cardinality of the set of possible input patterns X
y_j or y_{pj}	Response or output of node j to pattern p
$\bar{y}_{pj}$	Trace of the past activity of output y_{pj}

CHAPTER 1: INTRODUCTION

1.1. Problem Description

Foraging animals, computers learning to play games, and adaptive robot controllers actually have a great deal in common. Consider the following three examples:

- An animal, exploring, comes across two unfamiliar types of mushroom, one red and one yellow. The red is tasty and nutritious, while the yellow is tasty but causes indigestion. The animal, after only one or two meals, learns to continue eating red, but avoid yellow.
- A computer learning to play chess, without any instruction from a human programmer as to strategy, concludes after several successive losses that sacrificing a knight to take a pawn is self-defeating.
- A robot must accelerate to travel up a wedge-shaped incline, but after a few unsuccessful attempts, adjusts its program to decelerate rapidly at the top rather than be carried over the edge by momentum.

The most obvious similarity between these three learning systems is that all are adaptive learning systems which interact with their environment in a mutually dependent way, as shown in Figure 1.1.

An implication of this is that the adaptive system must keep pace with the environment – the animal must recognise and decide to stalk its prey before the prey wanders off, and the computer chess-player must decide on a move before its human opponent wanders off.

The next important similarity relating these three systems is that they learn by *reinforcement*, or signals from the outside world which serve to alter the behaviour of the system. In the case of the animal, reinforcement is taste or pain; for the

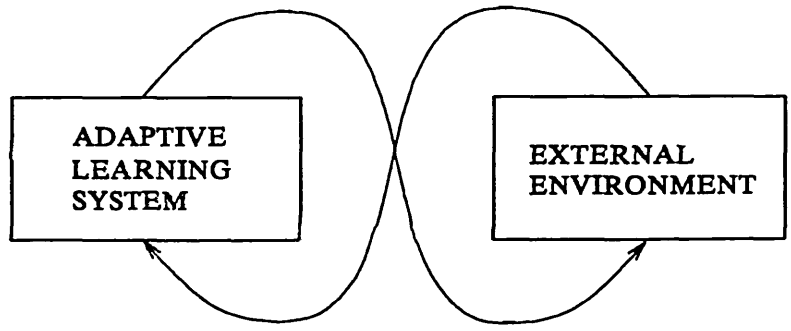


Figure 1.1. The systems of interest in this thesis are adaptive learning machines which receive input from the environment which is dependent in some way on their own previous outputs.

chess-player it is notification of win or loss; for the robot controller it may simply be sensor information that a servo has gone out of bounds, or it may be more dramatic sensory information following a crash. Each of these systems learns by trial-and-error: an action is selected and executed, and then the reinforcement associated with that action in the current environment is received. Learning is contingent on actions being made: no reinforcement will ever arrive if no responses are made. This type of learning is often termed *exploratory learning*, and is distinct from *passive learning*, where the system is shown a variety of pre-selected training examples and told how to respond to each.[1]

By its nature, this reinforcement is non-specific. The chess-player is told whether a move was a good one, but not what the optimal move would have been. This reinforcement may be binary (win/lose) or it may be an overall estimation of the goodness of the system response (whether the animal feels satiated or merely less hungry after consuming something). This paradigm is often called *reinforcement learning*. One alternative is *supervised learning*, in which the system response can

[1] Lawrence Hunter goes further to suggest that passive learning is not learning at all, in the usual sense of the word: "Learning, loosely stated, is the improvement of an organism's ability to achieve its goals on the basis of its experience. Clamping the input and the output of the system to a desired state is not what is traditionally meant by experience." [Hun88]

be compared with a "optimal" desired response. This presupposes that there is some meta-system available which knows the optimal responses already, and often this is simply not true. Another alternative learning paradigm is *unsupervised learning*, in which no reinforcement from the environment ever arrives, and the system is left to classify experiences into categories in any way it sees fit. This may be how areas of human visual cortex self-organise into line-detectors and the like, but is not appropriate for learning how to respond to an environment.

A final important commonality between the animal, chess-player and robot controller is that reinforcement to them only arrives at certain points – the tasting of the food after its acquisition, the end of the game, the failure of a control sequence, etc. This means that the chess-player, for example, receives one reinforcing signal which it must interpret in terms of a game which included perhaps several dozen system responses (moves). Worse, it is possible that other reinforcements, relating to other responses may arrive in the meantime. For example, the animal may have eaten several types of mushroom, and experienced the pleasurable taste reinforcement from each, by the time that the first pangs of indigestion are felt. This situation is often termed the *credit assignment problem*: determining which of previous system outputs are responsible for reinforcement. There are several levels on which this question may be addressed, of which the most abstract involves an understanding of structural cause-effect relationships, and as such it is beyond the scope of the simple systems considered here. Because of that, the term *credit assignment* is deliberately avoided in this thesis. Instead, the term used is taken from psychology – *delay learning* – and refers to associations formed even when results of actions are not available immediately, and when other actions and results may intervene.

1.2. Statement of Thesis Goal

According to the argument of the previous section, machines which solve this sort of learning problem must have (at a minimum) three basic properties:

P1. They must perform EXPLORATORY LEARNING – reinforcement from the external environment results from the system's own output, and this must be dealt with in an on-line manner rather than in batch learning mode.

P2. They must perform REINFORCEMENT LEARNING – the reinforcement from the environment does not take the form of providing the specific optimal output from each unit in the system. Rather, it will be a scalar indicating relative success or utility of the overall system output.

P3. They must be capable of DELAY LEARNING – the reinforcement resulting from a system output may not arrive immediately. Other system outputs may occur in the meantime, and other environmental reinforcements may also occur in the meantime.

This thesis presents a class of machine, consisting of Multi-valued Probabilistic Logic Nodes performing Attention-Driven Buffering, and shows that they can satisfy P1, P2 and P3.

1.3. Applicability of Neural Networks

The machines constructed in this thesis to satisfy P1, P2 and P3 are learning machines conforming to the artificial neural network (ANN) paradigm. Neural networks are composed of processing units which send and receive subsymbolic messages to one another, and which alter their behaviour as a result of a training rule which requires knowledge only of their own inputs and outputs.

One advantage of using ANNs, and the one of most concern within this thesis,

is that the systems find representations or rules governing their behaviour without provision of these by a programmer. For this reason, ANNs are often applied in domains where the rules are prohibitively complex or undefined. They are generally robust, meaning that the system can survive malfunction of some fraction of the processing units, which makes them suitable for real-world problems such as vehicle control. A neural network should also be able to handle some degree of noise in its inputs, and should be able to generalise from known information to new examples and even new concepts.

There are however a variety of learning paradigms besides ANNs, and therefore other ways in which properties P1 through P3 could doubtless be addressed. For example, one of the earliest and most famous learning systems, Samuel's checkers player [Sam63] is given a set of rules, and adapts the weightings of these rules to find an optimum strategy. Non-ANN strategies can be quite successful at learning particular tasks. It is worth noting that the system developed in Chapter 4 to satisfy P3 is actually sufficiently general to deal with *any* learning machine which satisfies P1 and P2, whether it be a neural network or not.

Having said that, it is still the case that ANNs provide a very natural way for addressing problems of exploratory reinforcement learning. They do not require, ideally, any pre-formed set of representations, but form representations automatically. It can even be argued that neural networks are more suitably used for this sort of task than for supervised passive learning, where :

> "...some agency must supply a rich set of training examples from which the required control can be formed. ... When it is available, such knowledge can be sufficient for one to implement the control rule in any number of different ways, many of which are easier than training an associative memory network." [Bar85]

ANNs start with a minimum of initial knowledge, and many can learn when the only information from the environment comes as a scalar reinforcement signal. Most other types of learning system require careful initialisation and require explicit

information on how to interpret external feedback and update their behaviour (c.f. [Sam63], [Dor68], [Boo88], etc.).

1.4. Overview of Solution and Results

The solution to the thesis goal has taken two distinct parts. The first is to design a neural network model which can satisfy P1 and P2, and the second is to use this model in a system which satisfies P3.

The neural network model designed is the Multi-valued Probabilistic Logic Node (MPLN), and is based on Aleksander's (3-valued) PLN. The PLN is a RAM-based node which learns by storing responses to input patterns in a lookup table, rather than by adjusting weights after the fashion of many conventional ANN models. Because it does not use weights, the PLN can often learn faster than other models. More to the point, it learns quite well with a training algorithm that implements reinforcement learning: the entire network updates on the basis of a binary reward/punish signal. This satisfies P1.

The MPLN allows storage of a range of values in the lookup table, each of which represent the probability of outputting a response, rather than only storing the response itself. This results in a node which can be trained quite easily in an incremental fashion, and in networks which can be trained on-line by a gradual shaping of behaviour. Because of this, MPLN networks perform well in exploratory learning situations: where the network makes a guess as to the appropriate response, is informed of the consequences, and alters its behaviour slightly in preparation for the next time that situation arises. MPLN networks therefore can satisfy P2.

MPLN networks can be used in systems which also satisfy P3. In particular, the systems which are developed in this thesis perform Attention-Driven Buffering (ADB). This involves keeping a trace or buffer at each node of some of its recent

input patterns. When reinforcement arrives, each node adjusts its response to each of those patterns accordingly and uniformly. The use of a measure of attention to each stored pattern ensures that those input patterns which are most unfamiliar or most unpredictable are most likely to remain in this buffer for the longest, and will thus be most likely to still be there when reinforcement arrives. In this way, learning is related to the predictions, or more accurately, to the inability of prediction, about the results of acting in response to an input state.

The chapters of this thesis which discuss the MPLN and ADB present some small experiments to justify the theoretical discussions of the systems. The principal results of this thesis take the form of simulations with OVSIM, a joint MPLN/ADB system, which learns discrimination tasks mimicking those to which *Octopus vulgaris* has been trained. The simulations show first, that OVSIM is capable of mastering similar tasks, and that several characteristics of its learning are similar to those observed in the animal as well. Second, after various functions in OVSIM are disabled, the system exhibits behavioural changes comparable to those seen in *Octopus* after damage of the brain regions believed to be responsible for delay learning. OVSIM may therefore have some claim to modelling the workings of the octopus brain.

1.5. Organisation of the Thesis

The remainder of the thesis is organised into five chapters. Chapter 2 first *deals* with the MPLN, covering its motivation, its precursor the 3-state PLN, and some simple experiments that show its utility as a neural network model. The MPLN is proposed as able to satisfy P1 and P2 simply.

The next two chapters are dedicated to systems exhibiting P3. Chapter 3 reviews related research. In Chapter 4, Attention-Driven Buffering (ADB) is introduced; its parameters are examined, and a small example is used to show that such a

system can learn when reinforcements are delayed and when contradictory signals arrive in the meantime.

Chapter 5 contains most of the experimental results in the thesis. It describes the octopus, an animal which can be trained to do operant learning which involves P1, P2 and P3. OVSIM, an MPLN/ADB system which simulates learning the same behaviours is described. The bulk of the chapter is concerned with experiments with OVSIM meant to mimic those in the octopus, and the similarities in patterns of learning and in the deficits which appear after similar sorts of damage to learning mechanisms.

Chapter 6 summarises the thesis, and argues that ADB may have some validity as a physiological model, particularly as the hippocampus and amygdala in mammals seem to be performing functions required by ADB. Future directions in which this work may proceed are also described.

CHAPTER 2 - PROBABILISTIC LOGIC NODES

2.1. Introduction

The problem posed in the previous chapter had three requirements: learning should be exploratory (P1), it should require only global feedback (P2), and this feedback should not need to be immediate (P3). If the solution is to take the form of a neural network, the first two requirements seem to be properties of the neural model and learning algorithm, while the third might logically be met at the level of the network architecture or control system.

Necessarily, the starting point for this thesis is a brief exposition of some artificial neural network (ANN) learning paradigms usually used in addressing problems such as P1, P2 and P3. Most of these center around the Weighted-Sum-and-Threshold model, which is most often trained via Error Back-propagation. Next, this model is compared with RAM-based models, and specifically with the Probabilistic Logic Node (PLN). Once this is done, the multi-valued PLN (MPLN) can be introduced and analysed as the first contribution of this thesis. It is claimed that networks of MPLNs can satisfy both P1 and P2 simply and naturally.

This chapter is also meant to prepare the way for the discussions which follow, by presenting accounts of the node models which are used in the systems described in Chapters 3 and 4.

2.2. Weighted-Sum-and-Threshold Models

In 1943, McCulloch and Pitts made an observation to which much ANN research can be traced [McP43]. They started with the common simplification that neuron output is "all-or-none" (firing or not firing), and went on to suggest that neurons could be treated as performing propositional logic on their inputs. Each neuron was normally quiescent (outputting binary 0) until a certain minimum number of its

inputs were activated, in which case it would fire (output binary 1). There were also inhibitory inputs whose activity could prevent the neuron from firing. McCulloch and Pitts showed that the behaviour of any network of these neurons could be described as a disjunction of logical minterms, and that any such proposition could be computed by some network.

These neurons are gross simplifications of real neurons (c.f., [CrA86], [Sej86], [Sel88]), but they have the advantage of being much easier to study and to construct. It is an open and controversial issue exactly how much biological realism may be sacrificed before the important information processing capabilities of real neurons are lost as well.

The common generalisation of McCulloch and Pitts's concept is the weighted-sum-and-threshold node shown in Figure 2.1.

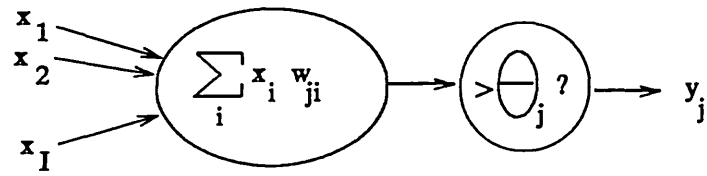


Figure 2.1. A weighted-sum-and-threshold node. Node j *computes the weighted sum of its inputs, and fires by outputting* 1 *if this sum exceeds some threshold* θ_j.

Node j receives inputs from some I sources. Associated with each input x_i is a weight, w_{ji}, and the node outputs according to the formula:

$$y_j = 1 \text{ iff } \sum_i x_i w_{ji} \geq \theta_j \tag{2.1}$$

where θ_j is node j's threshold.

In practice, the node is often designed to be capable of taking analog input and outputting an analog value, in which case the output function might be:

$$y_j = \sum_i x_i w_{ji} - \theta_j \tag{2.2}$$

Typically, the weights w_{ji} are restricted to a range [0,1] or [-1,1].

Nodes where binary or analog inputs are weighted and summed and compared with a threshold to determine a binary or analog output will be called Weighted-Sum-and-Threshold (WST) nodes.

There are numerous methods for training these nodes to perform desired input-output mappings. Many derive from the Hebbian formalism which is another simplification of observed biological behaviour: "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" [Heb49, p. 50]. Formally, if d_{pj} is the response desired from node j to pattern p, and x_{pi} is the i^{th} input to j from pattern p,

$$\Delta_p w_{ji} \propto d_{pj} x_{pi} \tag{2.3}$$

This rule strengthens weights between nodes that are frequently coactive. With repeated application, the node will learn to output $y_{pj}=d_{pj}$ for every pattern p, if the p are orthogonal. Orthogonality requires that no x_{pi} is activated by two patterns with differing d_{pj}. It does not encourage nodes to perform some desired mapping, as it makes no mention of the desired mapping – only of coactivity between nodes.

One useful learning rule, developed by Widrow and Hoff in 1960 [WiH60], and known variously as the Widrow-Hoff Rule, Delta Rule, or Least-Mean-Square (LMS) rule, simply adjusts the Hebbian rule to incorporate the node's current output y_{pj} as well as its desired or training output d_{pj}:

$$\Delta_p w_{ji} = \eta(d_{pj}-y_{pj})x_{pi} \tag{2.4}$$

With repeated applications, y_{pj} will approach d_{pj}, provided the input vectors are at least linearly separable. The training set is linearly separable if, when all I-dimensional input vectors are plotted in I-space, a single $I-1$-dimensional hyperplane can separate those patterns with $d_{pj}=0$ from those with $d_{pj}=1$.

The LMS rule is shown to perform gradient descent by Nagumo and Noda when $2>\eta>0$ [NaN67]. The proof defines the total error E over all training patterns:

$$E = \sum_p E_p, \text{ where } E_p = \frac{1}{2}\sum_j (d_{pj} - y_{pj})^2 \tag{2.5}$$

and then shows that E will decrease with each weight change according to 2.4.

The node defined by Equations 2.2 and 2.4 is often termed an ADALINE (ADAptive LINear Element, c.f. [WWB88]).

Rosenblatt's perceptrons are effectively supersets of the ADALINE [Ros58, Ros62]. Perceptrons are a class of adaptive node as shown in Figure 2.2, typically used for pattern recognition. The binary input pattern feeds into Association Units, which are hardwired to compute logical threshold functions of their inputs. These Association Units are often handcrafted feature detectors. The next layer contains Response Units, which are connected to the output of the Association Units via modifiable weights, and compute the weighted sum of their inputs. The Response Units are connected together in a winner-take-all fashion: that unit with the highest weighted sum is taken to "classify" the input as belonging to its class.

It will be noted from Figure 2.2 that the ADALINE is a perceptron where the Association Units are collapsed to have single-inputs and to perform the unity function, and where there is only a single Response Unit.

Perceptrons are trained according to a rule like Equation 2.4, where each Response Unit is treated as a separate node. Block, in the perceptron convergence theorem, showed that this training will terminate in finite time, and that the perceptron's response to each will be correct *given that the problem is* ~~*learnable*~~ performable *by the perceptron* [Blo62].

In fact, there are several important classes of problem which this perceptron will be unable to solve. Minsky and Papert showed that if the inputs to the

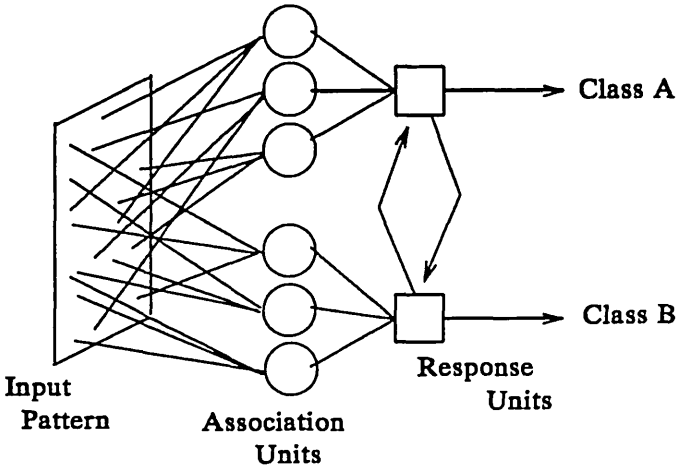


Figure 2.2. A typical perceptron. The binary input pattern feeds into fixed-logic Association Units, which in turn feed into Response Units via adaptive weights. The Response Unit with the highest weighted sum of its inputs "classifies" the input pattern.

Response Units are limited in number (i.e., no one node can see all of the input pattern), the perceptron will be unable to determine connectedness or number of regions in the pattern [MiP69]. Furthermore, they claimed (and still claim in the edition of their book which appeared in 1989) that even if a network of perceptron-like nodes can solve a particular problem, the solution found will not scale up. Finally, they made the important point that the perceptron learning algorithm performs gradient descent, and as such is susceptible to associated problems: namely, that the correct direction in the short term may not be correct in the long term, and the system may settle into a local solution rather than a global one.

Minsky and Papert levelled some criticisms against the capabilities of the perceptron which hold only in its simple form shown in Figure 2.2. If the nodes are arranged in multiple layers, so that the output from one node becomes input to the next level up, networks do exist which can be trained to perform classifications based on connectedness, parity, etc.

However, for a multi-layer network, Equation 2.4 is no longer sufficient as a

training rule. In a multi-layer network, d_{pj}, the desired response of each node j, is only available for those nodes at the highest level – output nodes. For the nodes at lower levels – hidden nodes – which pass output to nodes at higher levels, d_{pj} is not defined.

A solution to this problem was outlined by Rosenblatt [Ros62]: to propagate error backward from the visible output units to the hidden units which feed those output units and therefore helped to cause the error.

The Generalised Delta Rule, or Error Back-Propagation (EBP) was formalised independently by Rumelhart, Hinton and Williams [RHW86a,b], and several others ([Wer74], [Par85], [LeC86]). EBP allows the training of hidden nodes, but requires a feedforward topology – i.e., a node output must never be passed as input to nodes on the same or lower layers. It also requires a non-decreasing differentiable output function f_j, making the node output rule:

$$y_{pj} = f_j\left(\sum_i w_{ji} x_{pi}\right) \tag{2.6}$$

where x_{pi} may be external input or output from a node in the previous layer. Then if j is an output node (i.e., its desired response d_{pj} is defined),

$$\Delta w_{ji} = \eta\left(d_{pj} - y_{pj}\right) f'\left(\sum_i w_{ji} x_{pi}\right) x_{pi} \tag{2.7}$$

and if j is a hidden node,

$$\Delta w_{ji} = \eta \delta_{pj} x_{pi} \quad \textit{where} \quad \delta_{pj} = f'\left(\sum_i w_{ji} x_{pi}\right) \sum_k \delta_{pk} w_{kj} \tag{2.8}$$

To train such a net, weights are initialised to small random values. Each pattern p in the training set is applied and the actual change to w_{ji} is the net change resulting from the calculation of equation (2.7) or (2.8) on each pattern. This is repeated until error on the training set is eliminated or sufficiently small. Alternately, Δw_{ji} may be noted after each pattern is applied, and the net changes made to the weights after each complete presentation of the training set.

EBP guarantees a decrease in error with each weight change made. It performs gradient descent in error space, and as Minsky and Papert pointed out decades earlier [MiP69], this may be descent to a local minimum rather than to a true solution. Rumelhart, et. al. claim this is not a problem in practice [RHW86a].

There are several other difficulties with the EBP methodology.

- EBP is very slow. Even one of the smallest possible problems, learning to compute the exclusive-or of two input bits, in a net of two nodes (one hidden and one output) takes 558 sweeps through the four-pattern training set for a total of 2200 training steps [RHW86a]. Learning discrimination of a "T" and "C" pattern on a 3x3 grid, where the patterns may be rotated into any of four orientations, takes 5,000 to 10,000 presentations of each of the 8 patterns [RHW86a].

- Typically, each of the hidden nodes sees all of the input, while each output node sees all of the hidden layer output. This requires huge connectivity for any sizeable problem, and hence EBP nets often only seem practicable to construct in simulation.

- The error-assignment (Equations 2.7 and 2.8) requires a backward information flow, from output nodes back towards the input layer. As yet, there is no evidence that anything of this sort could occur in biological systems. Stork does show how a slightly more complex cluster of simple processing nodes can implement EBP in a more plausible way [Sto89]. Churchland and Sejnowski also note that while EBP can not be a literal model of learning at the neuronal level, it might be plausible as a systems-level description of learning in a net with feedback connections [ChS89].

- Parameters such as the learning constant, η, are problem-dependent and must generally be tuned empirically [ChV89]. A great deal of time at recent ANN conferences is devoted to work which centers on fine-tuning EBP to increase speed of learning (c.f., Proceedings IJCNN-90-WASH, 1990).

- EBP is restricted to feedforward networks: special adjustments must be made to handle feedforward and feedback loops as well as lateral inhibition on the same level

(c.f. [RHW86a], [WiZ89]).

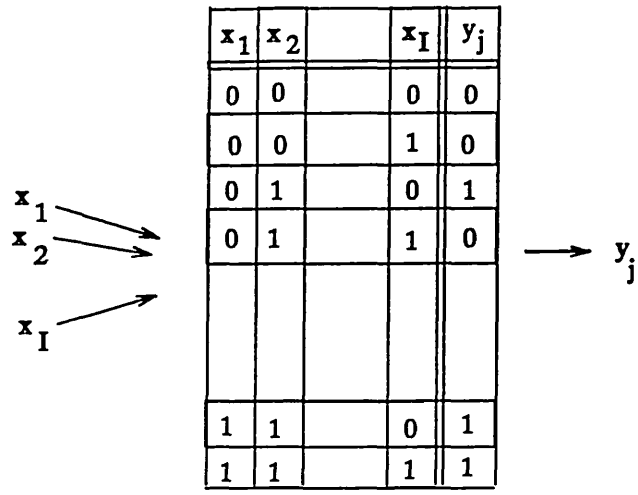
Nonetheless, EBP seems quite well-suited to many problems where the learning is off-line (and hence may take indefinitely long). One such problem is the learning of text-to-phoneme translation by Sejnowski and Rosenberg's NETtalk [SeR86]. EBP has also been applied with moderate success to such domains as speech recognition (c.f. [McE86]) and classification of sonar returns [GoS88].

There are other methods of training WST nodes within the supervised learning paradigm, but EBP and its variations are by far the most prevalent at present. There are also several methods for training WST nodes in unsupervised networks (e.g., Kohonen [Koh88], Grossberg [CaG87]). Unsupervised learning is not really appropriate for exploratory learning with delayed reinforcement – where the environment does provide some feedback. There are also methods for using WST nodes for reinforcement learning (e.g., Widrow [WiS64], Barto and Sutton [BSA83]). These methods are discussed in more detail in Chapter 3.

2.3. RAM-based Models

In the original specification of McCulloch and Pitts [McP43], no actual mention was made of weights on the inputs or of what changes should be made to the node to enact learning. Viewing nodes as executing logical propositions, it is equally valid to define learning as a change in the logical proposition executed, without any reference to weights. In this way, the node is more of a lookup table, in which a pattern on binary input lines addresses some location in the table, and the output of the node is dependent on a value stored at that location. A simple example of such a node is shown in Figure 2.3. Austin has shown how the paradigm may be extended to include grey-level (non-binary) inputs [Aus88].

In such a node, the logical proposition computed is the disjunction of all input



x_1	x_2		x_I	y_j
0	0		0	0
0	0		1	0
0	1		0	1
0	1		1	0
1	1		0	1
1	1		1	1

Figure 2.3. A logical or RAM-based node model. Inputs form an address into a lookup table memory, and the output of the node is the value stored at that location.

patterns for which the node will fire. The node has learned to respond correctly when it has stored the correct value for each possible input pattern at the associated address. An important feature of this kind of node is that no generalisation occurs: the node must store the appropriate response for every possible input. On the other hand, it can learn to execute any of the 2^{2^I} binary function of its I inputs, whereas the WST nodes are restricted to learn linearly separable functions.

The logical node traces its roots back to the n-tuple sampling machines of Bledsoe and Browning [BlB59], where the n inputs to the node form the n-tuple which is used to address node memory. Later, Aleksander and Stonham noted that the node memory addressing was analogous to that in a random-access memory (RAM), and therefore the nodes could be implemented in a straightforward way in existing technology [AlS79].

The WISARD pattern recognition machine, prototyped in 1981 [ATB84], makes use of these RAM-nodes in learning to classify images from a television camera. The binary image pixels are randomly assigned to n-tuples, each of which provides input to a RAM-node. Enough of these RAM-nodes so that the entire picture

is covered, plus a summator, form a *discriminator*. The machine contains one discriminator for each class of pattern to be recognised. The nodes in a discriminator are initialised to contain 0 at all locations; as each image to be learned is shown, the n-tuples address locations (one in each node) and these are set to contain 1. Then when an unknown image is shown, it addresses one location in each RAM-node, and the value found there is passed to the summator. If many of these addresses were accessed during training, the sum will be high. The discriminator with the highest sum wins, and the image is classified accordingly.

WISARD in one sense performs a simple Hamming distance comparison between trained and novel images. But it can also generalise, even though no single RAM-node is capable of generalisation itself. As a simple example, consider that a discriminator has been trained on two images, A and B, and is now shown a novel image U. U overlaps with A at a RAM-nodes, and therefore a 1's will be passed to the summator. But if U also overlaps with B at b nodes, and if some of these nodes are not the same as those overlapping with A, more 1's will be passed to the summator. Thus it is possible for the discriminator to give a higher response to U than would arise merely from its overlap with a single training pattern.

The WISARD has been implemented as a commercial machine, which can train on or classify 512x512 pixel binarised television images at a rate of 25 per second. Tattersall, et. al. also used a WISARD architecture (but the LMS training rule) to do speaker-independent recognition of utterances of the names of alphabetic characters [TFL89].

Recently, other workers have used RAM-nodes for a variety of tasks. Vidal has used RAM-nodes arranged in tree structures to do moving edge detection, tracking, and detecting patterns of some minimum size in the input data [Vid88]. Allinson, et. al. have used nodes with lookup tables to self-organise as an image classifier which, once trained, can classify video images at video speed [ABJ89]. Aleksander

and Wilson showed that RAM-nodes 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 [AlD85].

Patarnello and Carnevalli have trained RAM-node networks with simulated annealing – in which outputs are *initially* subject to a large degree of noise (the "temperature") which is gradually "cooled" to zero. This method is slow but provides one way of reducing the chances of settling into a false minimum. They kept the node contents fixed and optimised over the connections between nodes. Their systems can perform addition of two 8-bit numbers after exposure to only 0.003% of the possible examples [CaP87]. They also apply their learning method to a problem in which a "bug" learns to roam a simulated world in search of food [PaC90]. Inputs are provided by one sensor which fires if food is in the location directly ahead and a second which fires if food is in one of the other three immediately adjoining locations; the bug then outputs a decision to move ahead, turn left or right in place, or do nothing. Through training, the net learns a strategy such as: if the first sensor signals food directly ahead, move ahead; else if the second sensor is signalling, turn left; otherwise move ahead. Again, the learning regime is successful, but takes an extremely long time to accomplish its goal.

So, weightless RAM-nodes have had notable success in a number of applications. They also have several advantages over the weighted-sum-and-threshold approach. First, as the image and speech recognition systems cited show, learning can be very fast and even take place in real-time. Second, the nodes are implementable in currently available RAM technology. Third, RAM nodes often lend themselves to analysis by virtue of their logic basis. Fourth, a RAM node is able to implement *any* of the 2^{2^I} functions of its I inputs, whereas a weighted-sum-and-threshold node can only implement linearly-separable ones. Finally, the number of connections necessary tends to be low, certainly in comparison with weight-using

paradigms which are often either fully connected (e.g., [Hop82]) or else involve full connections between layers (e.g., [RHW86a]). This high interconnectivity poses a problem in VLSI design; optical implementations (c.f., [AbP87] and [PPH88]), which may be able to handle massive interconnectivity, are still themselves a research topic.

There is also a perceived advantage in the use of RAM nets as associative memory in terms of their storage capacity. Wong and Sherrington [WoS89] consider a randomly connected net of RAM nodes, each storing and outputting (-1,1) rather than (0,1), and define a cost function over the P patterns:

$$C_j = -\sum_{p=1}^{P} d_{pj} y_{pj} \tag{2.10}$$

They then show that this cost function is minimised by a "majority" learning rule: each storage location is assigned the value of the output required by the majority of the patterns addressing it. If a storage location is not addressed by any training pattern, it takes the value assigned its nearest neighbour in the lookup table – where neighbours are defined on the basis of close Hamming distance in address space.

Using this training scheme, Wong and Sherrington show that the storage capacity for associative retrieval in RAM nets scales as $2^I/I^2$. This compares quite favourably with weight-using nodes, arranged in the same topology, which can store a maximum of about $0.64I$ patterns [DGZ87], or about $0.15I$ if they are arranged in the standard Hopfield (fully-connected) topology (c.f. [Hop82], [New88], [AbS85]) in which every node receives input from every other node.

2.4. The Probabilistic Logic Node

The probabilistic logic node (PLN) represents an attempt to augment RAM nodes in such a way as to retain their desirable properties and improve upon them [KaA87, Ale88]. The PLN consists of a RAM-node augmented with a probabilistic

output generator. As in a simple RAM-node, the I binary inputs to a node form an address into one of the 2^I RAM locations. Simple RAM-nodes then output the stored value directly. In the PLN, the value stored at this address is passed through the output function which converts it into a binary node output. Thus each stored value may be interpreted as affecting the probability of outputting a 1 for a given pattern of node inputs. Figure 2.4 shows a PLN.

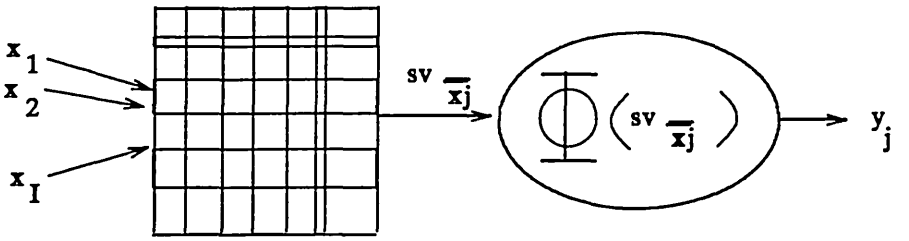


Figure 2.4. A probabilistic logic node (PLN). The node is a RAM-node augmented so that the value addressed by the current input is passed to a probabilistic transform Φ which converts it into the node's binary output.

In its most basic form, the alphabet Ω of stored values contains three elements: $\{0,1,u\}$, and the output function Φ is defined as:

$$\begin{aligned} &\Phi(1)=1 \\ &\Phi(0)=0 \\ &\Phi(u)=0,1 \textit{ with equal probability} \end{aligned} \tag{2.11}$$

This PLN may be implemented with RAM that addresses stored values occupying two-bit words. In this form of PLN, training becomes a process of replacing us with 0s and 1s so that the network consistently produces the correct output patterns in response to the training pattern inputs. At the start of training, all stored values in all nodes are initialised to u, and thus the net's behaviour is completely unbiased. In a fully converged PLN net, every addressed location should contain a 0 or a 1, and the net's behaviour will be completely deterministic.[1]

[1] There may be PLN locations which are never addressed, e.g. addresses to nodes in the input layer which represent n-tuples not present in the training set, or to nodes in higher layers if some combinations of lower node outputs never occur. These unaddressed locations may contain u without affecting the status of the net as converged. It is also possible for an unconverged network to behave deterministically, e.g., if all locations in all top layer nodes contain 1, but us remain in the lower levels. However, by

There are a number of training regimes for this type of network, of which the simplest and slowest is a random walk through possible combinations of stored value; a standard one is described by Aleksander [Ale88]. To train a PLN where ω, the number of elements in Ω, is 3 ($\{0,1,u\}$):

1. For all nodes j, for all addresses X, $sv_{Xj} = u$.
2. Choose an input pattern p.
3. Allow values to propagate through the net: each node forms an address $X=(x_1,x_2, \cdots x_I)$ from the values on its input lines, accesses a stored value sv_{Xj}, and outputs $y_{pj} = \Phi(sv_{Xj}) = 0$ or 1, according to the output rule in Equation 2.11.
4. When values appear at the output nodes, judge the output pattern "correct" ($r=+1$) or "incorrect" ($r=-1$).

4.1. If $r=+1$, do for each node j: If $sv_{Xj}=u$, $sv_{Xj} \leftarrow y_{pj}$. This ensures that current, successful behaviour will be repeated.

4.2. If $r=-1$, repeat from step 3., in hopes that some us may be transformed into different values which yield a more successful output.

4.3. If $r=-1$ after several applications of steps 3 and 4, a reset operation occurs.[2] For each node j: If $sv_{Xj} \neq u$, $sv_{Xj} \leftarrow u$. This returns the stored values to random.

5. Loop to step 2 until the network has achieved satisfactory responses to all patterns in the training set.

There are several important points to make about this algorithm:

- It is a reinforcement learning algorithm – in which the only reinforcement signal is a global scalar applied to the network as a whole. It does however involve the

Aleksander's training algorithm, this state is transient and all addressed locations will be set to 0/1 even if the setting is immaterial.

[2] If the net contains N nodes, and all were to address a u, there will be 2^N possible combinations of outputs $y_1 \cdots y_N$. This is an upper bound on the number of attempts to generate correct output which may be allowed.

network making several attempts to maximise this reinforcement signal (steps 4.2 and 4.3). Because no mechanism such as back-propagation is needed, the updates are simple, local, and may be executed in parallel.

- It makes no topological restrictions – the net may be feedforward, recurrent, fully connected, etc.

- Because learning is not done by gradient descent and instead has a probabilistic component, local minima are escapable, and learning may be expected to be faster on the problems such as those defined "hard" by Shapiro [Sha89a] where the error surface is flat and gradient descent degrades to random walk. [3]

- All memory locations are initialised to randomness, and thus the net starts off in a completely unbiased state. In contrast, in a weighted network, Kan and Aleksander [KaA87] note that the initial setting of the weights provides a pre-existing state structure which must be disrupted before training to the desired stable states can even begin. Learning in PLNs consists of, where possible, adjusting u values only – exploring uncommitted regions of function space. Only when the stored knowledge is *inconsistent* with the needs of the current training pattern does a "reset" routine disturb the stored knowledge, and return locations to the pool of unbiased memory.

In addition to this generic training algorithm, Aleksander has also defined a canonical form for PLN topologies [Ale89]. His canonical net is one in which all nodes have the same number of inputs, I; a pyramid topology is defined as a tree-like feedforward structure. Enough bottom-layer nodes are used to cover every bit

[3] Shapiro [Sha89a] considered 2-input simple RAM nodes, arranged in a tree structure, like that suggested by Aleksander (e.g., [Ale89]), and initialised to contain random 0s and 1s at each location. Then, defining the error as the proportion of patterns generating incorrect output, he trained by gradient descent: choosing a random location, calculate the error change if that value is flipped. If the error change is negative, the flip becomes permanent, if it is positive the value is left unchanged, and if there is no net error change, the location is set randomly. He found that the "hardness" of a problem, related to a low proportion of solution states among the possible states of a net, increased exponentially with the Minsky-Papert order – i.e., the number of inputs to a node required for the problem to be solvable.

in the input pattern once, and then additional layers are added to result in a single topmost output node. For a given size of the input pattern, there is only one parameter, I, needed to describe the topology. Enough pyramids are then used to generate the desired number of output bits, each of which may then be dependent on the entire input scene, as necessary.

Use of a topology such as the pyramid reduces the number of functions computable. If there are B bits in the input pattern, then there exist 2^{2^B} possible functions mapping them to a single output bit. However, in a net of N RAM nodes, there are at most 2^{N*2^I} implementable functions, and even then not all of them are distinct. For example, any function in which the topmost node is completely filled with 1s is exactly equivalent to any other function with the same event in the topmost node.

For the regular pyramidal topology, it is relatively straightforward to determine the number of functions computable for low I. Myers [Mye88] gives the number for an L-level pyramid of 2-input nodes as:

$$\begin{aligned} F(0,2) &= 4 \\ F(L,2) &= 2 + 2p + \frac{5}{2}p^2 \qquad (2.12) \\ &\quad \textit{where } p = F(L-1,2) - 2 \end{aligned}$$

Similarly, for an L-level pyramid of 3-input nodes, the number is given as:

$$\begin{aligned} F(0,3) &= 4 \\ F(L,3) &= 2 + 3p + \frac{15}{2}p^2 + \frac{108}{4}p^3 \qquad (2.13) \\ &\quad \textit{where } p = F(L-1,3) - 2 \end{aligned}$$

Al-Alawi and Stonham [AlS90] extend this to a network of nodes with arbitrary I.

While the use of such a topology dramatically reduces the number of functions executable by the system, as compared with a single, huge, B-input node, it is this very reduction which gives rise to generalisation. A single RAM-node cannot generalise, but a net of such nodes can no longer store a specific response to each input pattern, and thus must group similar patterns together in the same response class.

Shapiro [Sha89a] argues that the pyramid may be overly restraining as a topology. In a pyramid, there is only one path from any input bit to the single output node, yet of the 16 functions implementable by a 2-input node, only 10 are dependent on both inputs. If any node in the pyramid implements one of the other 6 functions, all of the inputs below it will be ignored. This suggests that a pyramidal topology will fare badly on higher-order problems – ones which depend on many inputs. He found that for the highest-order problem, the computation of parity, most changes to stored values result in no change in the error measure. Therefore, during much of training, the system is doing a random walk. Learning time in this case will be polynomial in the number of nodes but exponential in the number of bits in the input patterns. On the other hand, Aleksander's training algorithm differs from Shapiro's in that in the former several stored values are accessed simultaneously, and may be set or reset *en masse*. This attention to coherence between node settings may result in somewhat faster behaviour than Shapiro predicts.

Aleksander [Ale88] showed that a 3-node tree of 2-input PLNs, learning parity over a 4-bit input string, could converge in an average of 32 input presentations – or 2 sweeps through the training set. A net trained via EBP, with four hidden nodes and one output node, learned the same task in 2,825 cycles through the complete 16-pattern training set – i.e., some 45,000 input presentations [RHW86a]. Myers and Aleksander [MyA88] showed that PLN nets compared favourably with EBP nets in terms of speed on a variety of similar small problems.

Bisset, et. al. [BFF89] trained several nets of to recognise a training set of 300 machine-printed examples of each alphanumeric character[4] digitised and displayed on a 16x24 retina. Their PLN pyramids did not learn well, mainly because they used nodes with $I=2$, arranged in a 12-layer net. This arrangement allows overgen-

[4] The set consisted of 34 elements, A-Z, 0-9, with no distinction being made between I and 1 or between 0 and O.

eralisation and lack of specificity – while $2^{2^{384}} \approx 10^{3 \times 10^{114}}$ functions of the input exist, the network can implement only something on the order of $10^{3 \times 10^3}$ functions, a vanishingly small fraction of the possibilities. (In contrast, the EBP nets which Bisset, et. al. examined were constructed with 10 or 30 hidden nodes, each fully connected to all 384 input bits.) Still, the PLN nets in this experiment exhibited one quite desirable property: that of reaching peak performance within about one pass through the complete training set, even though this peak performance was low.

Other training algorithms for PLN nodes exist.

Wang and Grodin [WaG89] describe a training algorithm which is slightly less powerful than Aleksander's. Each stored value consists of two bits, the value to output and a flag indicating that that value should not be changed. Flags are set when correct output behaviour has been produced, so that the behaviour will be repeated. If incorrect output is generated, then a search is made of the unflagged locations in the net, and different stored values are tried. Once a flag is set, the value can never again be changed. This is the principal contrast with Aleksander's use of u, as the inability to error-correct makes Wang and Grodin's algorithm very dependent on both the initial output stored values and also the ordering of the training set.

Al-Alawi and Stonham [AlS90] describe a related algorithm which, on error, initiates a depth-first search of all u states currently accessed, and tries to find a successful solution by systematically trying all possible combinations of instantiations for those us. Only if the error still persists does the hard reset (step 4.3 in Aleksander's algorithm) occur. They give results for the parity problem which are only slightly worse than those in Aleksander [Ale88]: requiring three passes through the training set to Aleksander's two.

PLNs appear to learn more quickly than WST nodes trained via EBP, at least for topologies involving a small number of nodes. This is attributable first to the

linear independence of a RAM-node's stored values: in a weighted-sum-and-threshold net, changing a weight affects the node's response not only to the current input pattern, but to any number of others. A second reason for the gain in speed is the use of the u value to help restrict changes to the areas of function space where they will have the minimum possible effect on other previously stored data.

Aleksander [Ale90] has proposed an extension of the PLN, the gRAM, which once trained, spreads stored information to those neighbouring locations which still contain us. In this way, it is hoped that the nodes can be made less susceptible to noise (small variations in the input patterns which change the addresses to a node by one or two inputs) and also make them more able to generalise (by making the responses to patterns similar in address space more alike).

Even in their simplest, three-state forms, PLNs satisfy one of the three properties set forth in Chapter 1, P2: that learning occur with only a global scalar reinforcement signal. A slight extension of the PLN concept is developed in the remainder of this chapter and shown to immediately satisfy a second required property, P1: exploratory learning without a distinct training phase.

2.5. The Multivalued PLN

The simple PLN, with $\Omega=\{0,1,u\}$, trained with Aleksander's algorithm, performs a type of reinforcement learning: it makes use of a global scalar reinforcement signal. However, learning is not exploratory. The system should not enter the minor loop described in steps 3 and 4 of the algorithm, where it tries several times to generate correct output. Instead, the system should simply generate an output and experience the results.

Additionally, in Aleksander's algorithm, the network output is compared with the desired output, and in a true reinforcement learning paradigm, such desired out-

put would not be provided. The only information from the outside world should come via the reinforcement signal $r = \{-1,0,+1\}$.

However, these changes would also result in a system which was much more prone to experience resets. Reset in the PLN is a drastic affair: each node j will have its currently addressed stored value sv_{Xj} reset to u. Since each sv_{Xj} was changed from u in order to store a previous training pattern, the reset of these values will cause loss of information. Sometimes this will be unavoidable. But it is also possible that most of the sv_{Xj} ought not to be changed.

For a given network learning a solvable problem, there exist one or more *solutions* in function space, and the net is said to *converge* when it reaches one of them. At any point during training, the net will be closest to one (or equidistant from two or more). If the net is closest to just one, then there is a value sv_{Xj}^* for each stored value which is the value of sv_{Xj} at solution. (If there are two or more equidistant solutions, then either sv_{Xj}^* exists, being the same in each solution state, or else sv_{Xj}^* is undefined – meaning that a solution is equally close whichever value sv_{Xj} takes.) Then it is possible that at the time of generation of a reset, every stored value accessed is already set to sv_{Xj}^* except for one, sv_{ERRj}, and that that one sv_{ERRj} is sufficient to cause error at a network level. Then if there are N nodes in the net, $N-1$ stored values will be lost, while only 1 erroneous one is reset.

As an example [Mye87], 100 PLN networks, each containing 3 2-input nodes arranged in a tree, were required to learn to detect parity of the 4-bit input string. For this task, 4 of the 2^{12} possible network states were solution states, and during training the closest to any net could be determined. Each net was frozen after its first error generation (22% of the nets learned the problem without ever generating an error and were discarded from further analysis). The 78 nets each contained 3 sv_{Xj} due to be reset, for a total of 234 sv_{Xj}. Of these, for 53.42% $sv_{Xj} = sv_{Xj}^*$, and hence resetting these bits was actually a loss of valuable information. For only

20.09% did $sv_{Xj} \neq sv_{Xj}^*$ and therefore warrant reset. For the remaining 26.49%, sv_{Xj}^* was undefined. Thus in only about one-fifth of the cases was total reset actually warranted; in over one-half of the cases it actually served to erase necessary information.

In a network operating under Aleksander's training algorithm, this is apparently not very harmful, and the networks still learn quickly.

However, for an exploratory, reinforcement-learning system, such erasure of error is crippling: no sooner will the system have begun to approach one solution, than a minor error may divert it into a wildly different region of function space. If the algorithm is changed so that the minor loop (whereby the system tries to avoid reset) is eliminated, resets will become even more frequent, and the learning algorithm may well degrade to uninformed random exploration of function space.

Perhaps the obvious way to ease this situation is to make the reset operation less drastic. This is the goal behind the multivalued PLN (MPLN), first published in 1988 [MyA88], and the first contribution discussed in this thesis.

The MPLN is simply an extension of the PLN (henceforth the 3-state PLN) which allows $\omega > 3$, $\Omega = \{v_0, v_1, \cdots v_{\omega-1}\}$:

$$\Phi(v_i) = 1 \textit{ with probability } \Phi^P(v_i) \qquad 2.15$$

$$\Phi^P(v_i) = \frac{1}{1+e^{\alpha(-2\frac{i}{\omega-1}+1)}} \qquad 2.16$$

This describes an MPLN (or ω-state PLN) in which there are ω possible values (v_i) to store in each node location, and the node will output 1 with a probability related to the value's ordinality (i).

One result of extending the 3-state PLN to the MPLN is that the RAM-node locations may now store output probabilities which are more finely gradated than in the 3-state PLN – for example, it is possible that a node will output 1 with 3% probability under a certain input. This might be desirable in an exploratory system

which should occasionally deviate for the purposes of exploring.[5]

The second result of the extension is that Aleksander's learning algorithm may now be adapted to allow incremental changes in stored values. In this way, it is possible that no one reset erases much information, but that erroneous information is discarded only after a series of errors. Similarly, new information is only acquired after a series of experiences indicate its validity. Such a learning algorithm is the following [MyA88]:

1. For all nodes j, for all addresses X, $sv_{Xj}=v_u$, where v_u is the element of Ω such that $\Phi^P(v_u)=0.5$ (or nearest approximation if no such element exists).
2. Choose an input pattern p.
3. Allow values to propagate through the net: each node forms an address $X=(x_1,x_2,\cdots x_l)$ from the values on its input lines, accesses a stored value sv_{Xj}, and outputs $y_{pj} = \Phi(sv_{Xj}) = 0$ or 1. Φ^P denotes the probability that $y_{pj}=1$.
4. When values appear at the output nodes, judge the output pattern "correct" ($r=+1$) or "incorrect" ($r=-1$).

4.1. For each node j, with $sv_{Xj}=v_i$: Define $i'=\kappa_r*r$.

4.2. For each node j where $y_{pj}=1$: $sv_{Xj} \leftarrow v_{i+i'}$.

4.3. For each node j where $y_{pj}=0$: $sv_{Xj} \leftarrow v_{i-i'}$.

5. Loop to step 2. until the network has achieved satisfactory responses to all patterns in the training set.

It will be noticed that steps 4.2 and 4.3 allow $i+i'$ and $i-i'$ to grow beyond the range of 0..ω-1; this can be resolved simply by clipping.

This training algorithm involves a linear change to the stored values and a sig-

[5] The pRAM of Gorse and Taylor goes further to allow for continuous sv values; see discussion later in this section.

moidal output probability function such as that of Equation 2.16. Alternatively, if ω is large, it is possible to define a sigmoidal change to the stored values and a linear output function:

$$\Phi_2^P(v_i) = \frac{i}{\omega-1} \qquad 2.17$$

$$sv_{Xj}=v_{i'}, \quad i'_2 = \frac{1}{1+e^{\alpha(-2(\frac{x}{\omega-1})+1)}} \qquad 2.18$$

where $x=i+\kappa_r$ if $y_{pj}=1$ and $x=i-\kappa_r$ if $y_{pj}=0$.

These two cases are equivalent, since $\Phi^P(v_{i+i'})=\Phi_2^P(v_{i+i'_2})$. Therefore, any behaviour obtainable by a net with sigmoidal output function and linear stored value change is obtainable by a net with linear output function and sigmoidal stored value change, and vice versa. For the remainder of the thesis, then, only the case given by Equations 2.15 and 2.16 is considered.

An earlier report [MyA88] shows small networks of MPLNs trained on several problems such as detecting parity or symmetry of a string, encoding a string, and addition with carry of two 2-bit numbers. The MPLNs were found to learn faster than 3-state PLNs, which in turn took less time to train than the error back-propagation networks applied to the same problems by Rumelhart, et. al. [RHW86b].

Martland [Mar88] describes an MPLN training algorithm which is meant to mimic EBP. He considers each stored value *sv* as representing the probability that the node outputs a 1 when that value is addressed. Then, the learning rule can implement gradient descent:

$$\Delta sv = sv - \eta \partial Error/\partial sv \qquad (2.14)$$

where η is a learning constant. On problems of 4-bit and 8-bit parity, his regime performed about 100 times slower than that in [MyA88], but still faster by several times than an equivalent EBP learning algorithm operating on WST nodes.

Speed of learning and ease of implementation in hardware are characteristics of the PLN-type nodes which are important but not of primary concern to this thesis. It is more relevant that the MPLN, with the learning algorithm defined above, performs exploratory, reinforcement learning, and therefore satisfies both P1 and P2.

While the MPLN was being developed in 1987-1988, Taylor and Gorse were simultaneously and independently developing the pRAM (probabilistic RAM) model. The pRAM is also an extension of the PLN, and it is similar in many ways to the MPLN.

Taylor has developed a model of a noisy neuron (summarised in [GoT88]) which presents equations which incorporate and formalise many known properties of living neurons. Gorse and Taylor have shown the evolution of this model to be formally equivalent to that of a network of noisy RAMs or pRAM nodes [GoT88].

The pRAM in its simplest form is a lookup table in which each address stores a value $s \in [0,1]$; the pRAM will output a 1 (spike) with frequency related to the s addressed. Each pRAM receives NT inputs: showing the activities of the N neurons in the last T time steps. One output function is given in Clarkson, et. al. [CGT90]:

$$Prob(j\ fires | x_1 \cdots x_{NT}) = (1-y_j^t)\sum_{i=1}^{N}\sum_{r=1}^{T} K_{jir}(y_i^r \mu_{jir} + (1-y_i^r)\nu_{jir}) \qquad 2.19$$

y_j^r indicates the activity of the jth node at time $t-r$, where t is the current time; K_{jir} indicates the influence of the activity of the ith input of the jth node at time $t-r$. Then μ_{jir} (ν_{jir}) represents the expected contribution from deterministic (spontaneous) transmitter release into the synapse from i to j at time $t-r$.

The inputs to the pRAM can then be chosen so that a location is addressed which stores s, such that $s = Prob(j \text{ fires } | x_1 \cdots x_{NT})$.

A hardware implementation of a 2-node network of 2-input pRAMs has been constructed successfully [CGT89].

Recently, the pRAM output function has been simplified [GoT90]:

$$Prob(j\ fires|x_1 \cdots x_{NS}) = \sum_X s_X \prod_{i=1}^{N} (x_i X_i + (1-x_i)(1-X_i)) \qquad 2.20$$

where the $X=(X_1 \cdots X_{NT})$ are possible addresses into pRAM memory.

Alternatively, if the pRAM is augmented to contain an integrating register (the integrating- or i-pRAM), output can depend on activity over the last T time steps as [GoT90]:

$$Prob(j\ fires|recent\ input <\mathbf{x}>) = f\left(\sum_X w_{jX} \times Prob(X\ addressed\ in\ last\ T\ time\ steps) - \Theta\right) \qquad 2.21$$

f may be a sigmoid. If $T=1$ and $Prob(X\ addressed) \in \{0,1\}$, this is the MPLN model. With $T>1$ and with $Prob(X\ addressed) \in [0,1]$, the pRAM model is much more complex, and this is the principal difference with the MPLN: the pRAM can output frequencies truly dependent on the probabilities of accessing different s.

A secondary difference is that the stored values in the MPLN are selected from a finite range (cardinality ω), while those in the pRAM are continuously valued. But ω may be arbitrarily high, while in the implementation of a pRAM, finite word lengths will be used and so the values must be discretised (to $\omega=2^{32}$ for a 32-bit word length).

Section 2.6 suggests that, for the problems considered in this thesis, ω small ($5 \leq \omega \leq 15$) suffices. However, by using a theoretically continuous range, Taylor and Gorse can derive training rules for pRAMs from traditional supervised, unsupervised and reinforcement learning paradigms [GoT90]. Their reinforcement training rule is derived from Barto and Sutton and has the form:

$$\Delta s_X^j = \rho((y_j - w_X^j)r + \lambda((1-y_j) - w_X^j)p) \qquad 2.22$$

where w_X^j is the value stored in location X of node j, where ρ and λ are learning rates, and where r and p are the positive and negative reinforcement signals, respectively. Using this rule, Taylor and Gorse have matched some of the experiments

performed by Barto and Sutton [GoT90].

The work of Barto and Sutton is sufficiently related to this thesis to merit its own discussion in Chapter 3, and Gorse and Taylor's adaptation of it is mentioned further there in that context.

Meanwhile, much of the analysis of the MPLN in the next section may also be applied to the pRAM and to some extent to the 3-state PLN.

2.6. The Design of MPLNs

There are three variables to consider when instantiating an MPLN: I, the number of inputs to the node (and hence 2^I, the number of stored values in the node); Ω, the stored value alphabet, or merely ω, the cardinality of this alphabet; and Φ^P, the output probability function.

2.6.1. I - The number of inputs to a node

In any PLN, I has a direct influence on the memory requirements ($2^I \log_2 \omega$ is a measure of the size of the lookup table of the node), also on the tradeoff between generalisation and memorisation. For a single-layer network of 3-state PLNs, for example, if one pattern $P1$ has been trained, and the correct response d_{P1j} has been learned at node j, then if a new pattern U is presented, the probability of eliciting d_{P1j} is

$$Prob(y_{P1j}=d_{P1j}) = (A_{P1U})^I + \frac{1}{2}(1-(A_{P1U})^I) = \frac{1}{2}+\frac{1}{2}(A_{P1U})^I \qquad (2.23)$$

where A_{xy} is the proportion of pixels in common between patterns x and y – the overlap. Equation 2.23 is an immediate extension of the formula given in [AlD85] for deterministic RAM nodes, and it states that the probability that $y_{Uj}=y_{P1j}$ is equal to the probability that the n-tuple from U is sited entirely in the overlap between $P1$ and U plus the probability that it is not, but that the value addressed (u) is converted

into y_{P1j} anyway.

By extension of this (and by comparison with [AlD85]), if patterns $P1 \cdots Pk$ are trained to produce output 1 at node j, and if patterns $Q1 \cdots Ql$ are trained to output 0, then the probability that node j produces a 1 in response to U is:

$$Prob(y_{Uj}=1)=\sum_{i=1}^{k}(-1)^{i+1}PP(i) + \frac{1}{2}\left[1-\sum_{i=1}^{k}(-1)^{i+1}PP(i)\right]\left[1-\sum_{i=1}^{l}(-1)^{i+1}PQ(i)\right] \quad (2.24)$$

where $PP(x)$ is the probability that the n-tuple is sited in any overlap between x trained patterns and U:

$$PP(x)=\sum\left(A_{p1p2\ldots pxU}\right)^{I} \text{ for } all \left\{p1,p2,\cdots px\right\}\subseteq\left\{P1,P2,\cdots Pk\right\} \quad (2.25)$$

Similarly for PQ,

$$PQ(x)=\sum\left(A_{q1q2\ldots qxU}\right)^{I} \text{ for } all \left\{q1,q2,\cdots qx\right\}\subseteq\left\{Q1,Q2,\cdots Ql\right\} \quad (2.26)$$

Defining generalisation in terms of the likelihood of outputting to pattern U in the same way as to a similar trained pattern, it will be seen that for a given training set, the generalisation of this (single-layer) net is wholly dependent on I. In particular, generalisation decreases as I increases, while specificity (ability to memorise responses to larger or more similar training sets) increases with I. This is true for a 3-state PLN; for an MPLN, the $\frac{1}{2}$ term in equation (2.24) becomes dependent on Φ^P, but the dependence on I remains. Similarly, as more levels are added to the network, the formulae become more complex, but the basic dependence on I remains.[6]

[6] Kan [Kan88] has designed a 3-state PLN system to maintain low I, and hence low memory requirements, but increase specificity and therefore memory capacity. He achieves this by transforming non-orthogonal inputs to address a different node location for each different output required. No disruption of previously trained states by new conflicting ones then occurs, since every new write is only made to a location not previously addressed. In specific, he amplifies the Hamming distance between patterns, using layers of nodes which repeatedly generate new addresses after a fashion not unlike hashing. After a few layers, similar inputs become quite differentiated in Hamming terms, and thus can be stored without conflict.

2.6.2. Φ^P — The output probability function

The output function, Φ, is a rule by which the node determines its output, given a certain pattern on its input lines. It is governed, for a non-deterministic node, by the output probability function Φ^P: $Prob(\Phi(x)=1) = \Phi^P(x)$. In the case of PLNs, it is the mechanism whereby stored values are interpreted as affecting the probability that a 1 is output at that node. In particular, $\Phi^P(v_i)$ is the probability that $\Phi(v_i)=1$.

In equational form, the output of a 3-state PLN is determined by:

$$\begin{aligned} \Phi^P(v_0)&=\Phi^P(0)=0 \\ \Phi^P(v_1)&=\Phi^P(u)=0.5 \\ \Phi^P(v_2)&=\Phi^P(1)=1.0 \end{aligned} \tag{2.27}$$

or simply, $\Phi^P(v_i)=\frac{i}{2}$.

In an MPLN, typically one element of Ω, v_u, assumes the role of u, so that $\Phi(v_u)=1$ with probability of 1/2. Two possible Φ^P are:

$$\Phi^P_S(v_i)=\frac{i}{\omega-1} \tag{2.28}$$

$$\begin{aligned} \Phi^P_H(v_i)&=1, \text{ if } i>u \\ &=0.5 \text{ if } i=u \\ &=0, \text{ if } i<u \end{aligned} \tag{2.29}$$

Φ^P_S is a step function approximating a linear output function. Because of the mutual independence of the stored values sv_{Xj} within a single node, the MPLN is not restricted to linear functions, and may execute any arbitrary (non-monotonic, non-smooth, non-differentiable, etc.) output function. Φ^P_H is a step function which approximates a very steep sigmoidal curve. These two, Φ^P_S and Φ^P_H, may be viewed as creating maximally different output functions: the first is very "soft" — if $i1 \approx i2$, $\Phi^P(i1) \approx \Phi^P(i2)$; the second is "hard" — $\Phi^P(v_{u-1})$ is maximally different from $\Phi^P(v_{u+1})$. Of course there exist an infinite number of curves between these two

extremes which Φ^P may approximate.

A likely criterion for choice of Φ^P is its effect on the speed of convergence of the network. Myers [Mye89a] shows theoretically and experimentally that speed can be maximised, under certain conditions, when Φ^P approximates a very steep sigmoid. Wong and Sherrington also show that with a steep Φ^P, and with $\omega=11$, the net is much more robust with respect to training noise than it is at lower ω, and also that there is a greater storage capacity [WoS89].

The result is intuitively satisfying: it suggests that once a node location is "committed" to an output, i.e., that it has been reinforced even once away from v_u and toward 0 or 1, it should output that value consistently. This allows other locations in the net to organise around one another with some confidence that all are behaving as they expect to behave when fully trained.

It is also possible to anneal an MPLN system. This would involve "cooling" the system by gradually changing from a linear Φ^P to a steep one. This has the same effect as changing from a high temperature (high noise) to a low one (low noise) in a WST system. It is to be expected that this strategy, like for WST nodes, would help avoid the system settling into local minima, but that it would take an extremely long convergence time.

2.6.3. ω - The cardinality of the stored value alphabet

The final major variable in the MPLN is Ω, or simply ω, the cardinality of Ω.

The advantage of $\omega>3$ (and thus of the MPLN over the 3-state PLN) is that after several reinforcements of a sv_{Xj} in one direction – so sv_{Xj} stores v_i, $i\approx 0$ or $i\approx\omega-1$ it can be hard to erase that knowledge: in fact, it will take an equal number of negative experiences to return it to v_u. Errors may occur due to mis-set stored values elsewhere in the net, but during which sv_{Xj} happens to be addressed, and sv_{Xj}

will be negatively reinforced since the error signal is global and indiscriminate. If ω is high, and v_i is far from v_u, sv_{Xj} will be pushed back towards v_u but only by $\frac{1}{\omega}$, and the probability of outputting a 1 when sv_{Xj} is addressed need only change very little – particularly if Φ^P is a steep sigmoid. In a network where $\omega=3$, in contrast, as discussed above, a single error arising anywhere in the net results in one location in each node being reset to v_u, and thus a great deal of knowledge is erased, regardless of whether any individual node is responsible for the error.

Increasing ω has its costs. The first is that an order of $\log_2\omega$ bits will be required to store each $v_i \in \Omega$, making the RAM needed in each node scale as $2^I \log_2\omega$. The equation is exponential in I, the number of inputs to the node, and this may dominate the cost for moderate ω.

Also, with a sv_{Xj} which stores v_i, $i \approx 0$ or $i \approx \omega-1$, high ω means that it will be very difficult to return sv_{Xj} to v_u when this is required. Noisy data or an unfortunate ordering of training examples could push a location's value very far from v_u and away from sv_{Xj}^*, and an equal number of error cycles will be required to reset it.

Kohring [Koh90] has shown that for WST nodes also, storage is more efficient (by up to 25 times) when synapses have a finite number of states than when they are continuously ranging. He makes the point: "... given the noisy character of the individual neurons, it is implausible that an arbitrarily small change in the neuron firing rate is meaningful. Rather, if two different firing rates are to be significant, then the difference between the two firing rates must be greater than that resulting from noise. ...this naturally leads to the concept of neurons with only a finite number of firing states." In an MPLN, the firing rate is derived from the stored values, and hence these also have a limited number ω.

Ideally, ω must be chosen to balance protection against mistaken erasure versus ability to erase when this is necessary. Myers [Mye89a] indicates that a ω larger than 3, but still relatively small, leads to good convergence speed: in particular, the

rule $5 \leq \omega \leq 15$ is suggested. Experiments reported there support this result: for example on several problems nets with $\omega = 11$ converge faster than nets with $\omega = 6$ or $\omega = 20$.

2.6.4. MPLN Parameters - Conclusions

Several assumptions are implicit in the choice of Φ_H^P and $\omega \approx 11$. The analyses consider feedforward pyramids, being trained on problems for which convergence is possible, via a training schedule which involves a random ordering of training patterns. This is a constrained class of topology and task, but one which is still quite powerful.

Given these assumptions, an MPLN net may be designed which will tend to converge as fast as possible: namely, its nodes contain stored values selected from a 5-15 element alphabet, and which are interpreted according to a threshold-like output function. The experiments described in support of these claims (in [Mye89a]) are small both in terms of the number of nodes involved and also in terms of the size of state space relative to the number of solutions available. They are useful however since a small number of distinguishable solutions exist and since the parity problem in particular is arguably the "hardest" of the hard learning problems.

It is not the case that the ω and Φ^P defined in [Mye89a] are universally optimal; it is not clear in the first place that speed of convergence is a necessary criterion to judge the "success" of a network – although it is probably the most frequent. There are occasions when a soft output function, for example, will be desirable despite its slowness. One obvious example involves a state space with abundant and deep local minima (false solutions), where probabilistic noisy outputs are necessary; in effect, a network using a steep output function forms quick and binding opinions, whereas a network with a more linear output function makes conservative ones, which still allow occasional lapses into the opposite output. This ability would

prove important in simulation of an automaton existing in a changing environment, where convergence *per se* is not possible, and where a net might be more successful if some of its nodes, say, output a 1 most of the time, and occasionally output a 0 to test the effects in the current environment.

Appropriate choice of parameters is therefore highly dependent on the size, shape and complexity of the problem space, and also causes subtle changes in the way the net organises to solve the problem – particularly in terms of the speed with which nodes commit to a particular output in response to some address. No values of ω and Φ^P can therefore be purported to be optimal under all conditions, merely as especially useful, and as good first approximations for later fine-tuning as necessary.

In later chapters, $\omega = 11$ and Φ^P a steep sigmoid are not always used, as different problem classes require different design strategies. But the analysis of I, ω and Φ^P (here and in [Mye89a]) allows informed selection of these parameters to suit the problems under consideration.

2.7. Summary of Chapter 2

This chapter covered some ANN paradigms including the Probabilistic Logic Node (PLN), a RAM-based node with a probabilistic operator which transforms stored values into probabilities of firing. This type of node is able to learn any boolean function of its inputs, its use in a pyramidal topology requires only low interconnectivity between nodes, and it has exhibited faster learning than nets trained with error back-propagation, on some small problems.

Sections 2.5 and 2.6 contain the main theoretical contributions of this chapter. First, the multi-valued PLN (MPLN) was introduced and discussed; MPLNs have shown faster learning than 3-state PLNs, and also allow for an incremental rein-

forcement learning algorithm. Use of MPLNs introduces several additional variable into the network design phase, as compared with the use of PLNs. These variables include the alphabet of possible stored values (cardinality ω), the function for updating stored values upon reinforcement, and the output probability function to convert stored values into firing probabilities (Φ^P). It was shown that judicious choice of Φ^P allows the stored value update function to be simplified to a linear function – so that the values can be changed by a constant amount κ_r on receipt of reinforcement r. The rule relating I, the number of inputs per node, to network generalisation capability of a network of simple RAMs was extended to the PLN case for single layer nets; and it was mentioned how this rule could be further extended to MPLN networks of arbitrary depth. Finally, it has been shown by analysis and experimentation that use of a steep sigmoid for the output probability function and use of a stored value alphabet with $5 \leq \omega \leq 15$ can speed learning.

MPLN networks with the training algorithm listed in Section 2.5 allow exploratory reinforcement learning.

It remains to show how a system involving a network of MPLNs can also be made to satisfy P3 and perform delay learning. The next chapters describe how some alternative approaches to this task have been made, and examine a system based on an MPLN network which satisfies P3 as well as P1 and P2.

CHAPTER 3 – RELATED LEARNING SYSTEMS

This chapter reviews a selection of artificial learning systems which exhibit at least one of the properties P1, P2 and P3. There is considerable overlap – for instance, many systems which perform delay learning are within the reinforcement learning paradigm. The final section of this chapter discusses features of this previous work which are incorporated into the system presented in the next chapter.

Some rule-based approaches have been included in this survey for completeness, but systems may be assumed to be ANN-based unless otherwise specified.

3.1. P1: Exploratory Learning

Exploratory learning was defined in Chapter 1 as involving an automaton interacting with an environment. The automaton selects and executes an action, and the environment provides reinforcement or feedback based on this action. In the opposite situation, *passive learning*, the automaton is repeatedly shown sample input/output pairs from a preselected training set.

There are basically two ways in which exploratory learning can be performed.

The automaton can *learn by trial and error*. The system responds to an input, and the response is then compared with the desired or predicted response. Reinforcement is dependent on the difference between actual and desired responses.

Alternatively, the system can *learn by doing*. Here, the automaton simply responds to inputs and observes what results follow – but no supervisor exists to provide the desired output for comparison.

3.1.1. Learning by trial and error

Trial and error learning is frequently used in adaptive control of robot arms

(e.g., [KuR89], [MKS88], [deC86]). Higher level control, or a random pattern generator, selects a desired target position. The system produces commands to move the arm, and sensors return information about the actual arm position reached. Error is calculated as the difference between target and actual position, and the system is trained to minimise this error by an algorithm such as error back-propagation.

In a control system, the output must pass through a complex "plant" or actuator before the commands affect the environment. In this case, the visible output from the plant is not connected in a simple way with the desired output from the control system. Following Psaltis, et. al. [PSY87] and treating the plant as a fixed layer, error may be backpropagated through the plant to derive error at the output of the control system, and then the control system may be trained using this derived error. Saerens and Soquet use this technique to train an exploratory ANS to drive simulated 2-dimensional arms or windows tracking moving targets [SaS89]. Their system also successfully learns the inverted pendulum problem: where a hinged pole rests atop a cart which moves along a finite one-dimensional track. The task is to balance the pole upright by small left and right movements of the cart on the track.

Shepanski and Macy have built a system to control a car moving around a simulated racetrack, which must keep safe distances from cars ahead of it, and which must learn when and how to change lanes and pass other cars [ShM88]. This system is trained in several stages; exploratory learning is used to teach how to follow a pace car which varies its speed at random. The system can output a decision to maintain current speed or to increment or decrement its own speed, and is trained by error backpropagation, where the desired output is one which results in the optimal following distance to the car ahead. After some 1,000 steps, the system maintains this optimal following distance and responds quickly to changes in the speed of the pace car.

Another use of multi-stage training occurs in Nguyen and Widrow's truck

backer-upper, which learns to back a simulated cab-trailer rig so that the rear of the trailer is aligned to the dock [NgW89]. The system is first taught the kinematics of cab-trailer interaction by error backpropagation. Given the location and angle of the cab and the trailer with respect to the loading dock and a steering signal (left or right), it learns to predict the resulting new locations and angles. Next the system learns by trial and error how to back the rig into the desired location opposite the loading dock. The truck backs under system control until it hits the loading dock wall. An error signal is generated based on how far the rear of the trailer is from the dock, and this is used to train each of the moves in the sequence by error back-propagation. When learning is complete, the system can learn to manoeuver the truck successfully even if the starting position is jackknifed.

3.1.2. Learning by doing

Learning by doing refers to exploratory learning situations in which no desired output is provided. It is more appropriate than learning by trial and error in situations in which the target is not defined or not known.

An early system of this type was built by Widrow and Smith and learned the inverted pendulum problem [WiS64]. It consisted of an ADALINE which received information about the pendulum angle and velocity and about the cart location and velocity, and output a decision to move the cart left or right. When a human observer judged that a recent sequence had been notably better (worse) than usual, a positive (negative) reinforcement was generated, and applied to each move in the sequence. The optimal sequence – or the amount by which the actual sequence deviated from optimal – was not provided.

A frequent task put to ANNs which learn by doing is that of simulating the movements of creatures seeking food in a two-dimensional world. Usually, the world consists of a grid, with some moves that contain food, and the creature con-

trolled by the learning system moves from cell to cell "eating" whenever it enters a food-containing cell. The goal is for the system to maximise the frequency with which it finds food.

Cecconi and Parisi designed a such a creature [CeP89]. It can output a decision to turn to the left or right, to move one cell forward, or to do nothing; its inputs consist of the distance to and angle with the nearest food. The creature initially wanders at random; when it happens to enter a cell containing food, the system stops and is returned to its state of 8 time steps previously. Then it wanders again, but at each step the previous action at that step is used as a training signal – since that previous action led to food. In this way, the system creates its own desired output patterns, and can be trained via error backpropagation, even in the absence of desired output provided by the environment. This system has been shown quite successful in learning to maximise the amount of food found.

Another food-seeking creature was constructed by Patarnello and Carnevalli [PaC90]. It differs from the other systems described so far in that it consists of fixed-function RAM nodes, and optimises over the connections between them. Its outputs are the same as in Cecconi and Parisi's system – turn right or left, move ahead, do nothing – but its inputs are much more restricted. It has only two sensors, one which fires if the creature is next to and facing a cell which contains food, and one which fires if there is food in any of the other neighbouring cells. The creature is trained by simulated annealing (as described in Section 2.3). Several worlds are generated with the same characteristics (e.g., a high density of food near the center), and the creature is allowed to live for a while in each. The amount of food eaten is then taken as the variable to be optimised.

Doran's rule-based system simulates a creature which – rather than seeking food – learns to find its way to a burrow in a static maze-like world [Dor68]. The creature wanders, keeping a detailed history of each move it makes. When placed in

a new position, after training, it performs a minimaxing operation on this information to generate an efficient path back to its burrow.

The environment confronting Ackley and Littman's simulated creature is complex: containing multiple autonomous creatures, ambulatory predators, impassable walls, and sheltering trees (to hide from predators) as well as food objects [AcL90]. Their creatures receive input about objects up to 4 cells away in each of the four directions; the creatures explore the world, and are trained by a reinforcement learning algorithm to maximise food (and minimise damage from hitting walls or from attacks by predators). Creatures are then spawned by genetic evolution: a creature which learns well is more likely to pass on its genes (as represented by its start state) than one which does not learn so well. After several generations, some creatures managed to survive for more than one million time steps.

The MPLN learning algorithm presented in Section 2.5 also performs exploratory learning. In it, the network selects an output, and this output is evaluated with respect to the environment: the optimal output is not provided, merely its appropriateness. If the environment is one which depends on the MPLN output, e.g., if the problem being learned involves inverted pendulums or food-seeking creatures or motor control, then the MPLN network will learn by doing.

3.1.3. Evaluation of exploratory learning techniques

The first type of exploratory learning, by trial and error, is very close to passive learning in the sense that the desired output is still provided, and the training regime is usually supervised learning (e.g., EBP). The explicit provision of a desired output means that these systems violate P2, reinforcement learning, and therefore are not suitable for the purposes of this thesis.

The systems which learn by doing are much more appropriate to this thesis.

They do not involve use of an externally-provided desired output pattern, only a global error signal, and therefore they to some extent perform reinforcement learning. Further, the systems described in Section 3.2.2 all receive this error signal only at the end of some sequence, and they must then recall their previous outputs and associate them with the error signal. This is of course delay learning (P3). However, most must keep long histories of past actions. The clearest example of this is Doran's system which maintains a history of all of its past actions.

3.2. P2: Reinforcement Learning

The defining characteristic of *reinforcement learning* is that there is no provision of an optimal or desired output pattern, merely an evaluation of the output pattern produced by the learning system. In *supervised learning*, this desired output is provided and the system learns to mimic it; while in *unsupervised learning*, there is no feedback from the external environment at all.

Reinforcement learning does not necessarily entail exploratory learning, and there are reinforcement systems which learn simply by passively noting the input/output training pairs shown to them.

Two sorts of reinforcement learning are covered here: global reinforcement systems, in which network weights are changed proportionally to the global error signal, and systems which use the global error signal to construct a desired output pattern, which the net then learns to mimic.

3.2.1. Learning with global reinforcement signals

The simplest type of reinforcement learning system is one in which the environment supplies a reinforcement signal r ($r=+1$ for reward, $r=-1$ for punishment, and possibly $r=0$ for no reinforcement), and each active weight is changed as a function of r:

$$\Delta w_{ji} = \alpha x_i y_j r \qquad 3.1$$

It is worth noting that when the networks involved consist only of a single node with binary output, this error signal is exactly equal to the desired output of that node, and so the learning is really supervised. For this reason, it only makes sense to talk about reinforcement learning in the context of a network of multiple nodes or when the same reinforcement signal is applied to each element of a sequence of actions.

Several systems already mentioned employ this technique: the inverted pendulum balancer of Widrow and Smith (Section 3.1.2 [WiS64]), the food-seeking creature of Cecconi and Parisi (Section 3.1.2 [CeP89]), and the MPLN learning algorithm (Section 2.5).

Windecker uses such a system to learn to play matching pennies [Win88]: the system tries to match the output of its opponent ("heads"=1, "tails"=0), who is assumed to play a fixed strategy. The system selects the output 1 when at least half of its nodes output a 1. When the system chooses the same output as the opponent, it wins and $r=+1$; otherwise $r=-1$. Windecker notes that the network quickly adapts so that at least half of the nodes output the correct (winning) value most of the time, and so the system generally wins. But once winning becomes frequent, there will be a small population of nodes which are usually wrong, but which are rewarded due to the overall system win. His strategy is to recognise this phenomenon and train this leftover population explicitly (which is of course recourse to supervised learning).

The problem is also noted by Widrow et. al [WGM73]. In a system in which there is some element of chance or noise, such as a game, the optimal decision under certain conditions is not always guaranteed to lead to positive reinforcement. For example, in Widrow et. al's blackjack player [WGM73], there is an optimal strategy which involves heuristics about when to request another card; yet this optimal strategy may not always win, and a non-optimal strategy may sometimes win simply by

good luck. In this way, r may be a source of confusion. The blackjack playing system learns by a rule similar to Equation 3.1, applied at the end of a game to all input/action pairs encountered during that game. Particularly when the effects of reward are stronger than those of punishment, the system approaches the performance of the optimal system after playing a few thousand games.

However, in any approach of this sort, even without the addition of chance or noise, reinforcement learning takes considerably longer than supervised learning because the reinforcement to any one node is proportional to the system response, rather than to its own output.

The BOXES system of Michie and Chambers [MiC68] works along similar lines. It contains one "box" or demon for each state. In effect, the system is a large ADALINE with one weighted input line associated with each possible state. It is also equivalent to a single large RAM node: a box is addressed by the input pattern in the same way as are stored values in a RAM node. As each box is entered, it chooses the system output and records its choice. When reinforcement arrives, each recently-entered box adjusts the probability of repeating its last output according to whether it participated in a successful sequence.

BOXES was applied to several problems including tic-tac-toe: each box recorded, for each possible next move, the number of subsequent wins and losses when that move was selected in the past. It then selected the move which most frequently led to wins. This strategy works well for tic-tac-toe, which has a large but not unimaginable number of possible board states; it would be nearly impossible to apply it to a task where the number of states is immense, such as chess.

The tic-tac-toe strategy is also inadequate for the inverted pendulum task; as all left-right moves eventually lead to failure, each box would find the number of failures after selecting an output exactly equal to how many times that output was selected. Instead of this failure statistic, BOXES stored the expected life for each

move – that is, it selected the move which, in the past, was followed by a greater time until the crash occurred. It updated by punishing each move made on a trial, but with decaying eligibility, so later choices were punished most. In this experiment, input was highly quantised, and so the number of input states was only 225; for problems with greater input space, BOXES quickly becomes impractical.

Some of the most famous work in reinforcement learning has been done by Barto and Sutton and their colleagues. One learning rule which they have produced is the Adaptive Reward-Penalty (A_{R-P}) algorithm, which is characterised by the following equations [BaA85]:

$$y_j = 1 \text{ if } \sum_{i=1}^{I} w_{ji}x_i + \eta > 0 \qquad 3.2$$

$$p_{ji} = Prob(\sum_{i=1}^{I} w_{ji}x_i + \eta > 0) \qquad 3.3$$

$$\Delta w_{ji} = \begin{cases} \rho(y_j - p_{ji})x_i & \text{if } r = 1 \\ \lambda\rho(1 - y_j - p_{ji})x_i & \text{if } r = 0 \end{cases} \qquad 3.4$$

By Equation 3.2, the A_{R-P} node is a WST node with random added noise η; Equation 3.3 defines the probability that the node output y_j will be 1. The weight change rule, 3.4, causes then node to be more likely to produce y_j when next the input x_i is presented, if the reinforcement r is positive, and less likely if it is negative.

Barto and Sutton have used a network of these elements to learn to behave as a food-finding creature [BaS81]. The network topology, as shown in Figure 3.1, consists of several nodes and a single reinforcement signal. The output rule for these nodes is:

$$\begin{aligned} p_{ji} &= 0.5 \text{ if } \sum_{i=1}^{I} w_{ji}x_i = 0 \\ p_{ji} &> 0.5 \text{ if } \sum_{i=1}^{I} w_{ji}x_i > 0 \\ p_{ji} &< 0.5 \text{ if } \sum_{i=1}^{I} w_{ji}x_i < 0 \end{aligned} \qquad 3.5$$

The environment in this problem consists of a two-dimensional world containing 4 landmarks, each of which emit a distinct odour which decreases in strength with

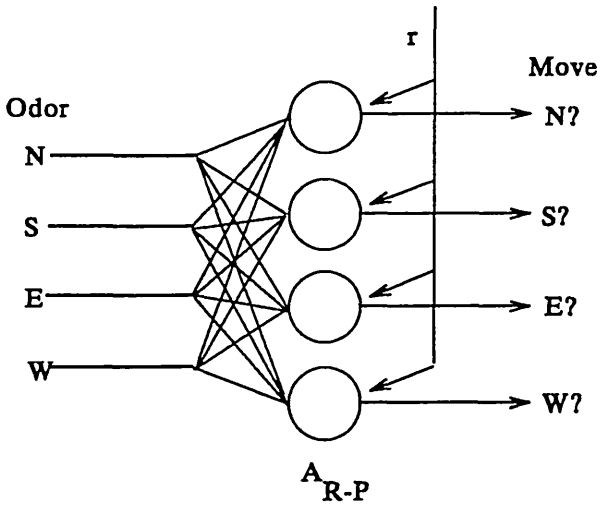


Figure 3.1. A network of A_{R-P} units used as a food-finding creature [BaS81].

distance. The strengths of these four smells serve as the input to the automaton. The automaton has four output lines, each of which signal that the creature should move in one of the four compass directions (combination moves, such as north-east are possible). There is a fifth, attractant location, which also emits an odour; the creature receives reinforcement proportional to changes in the strength of the attractant smell.

The task of this creature is to maximise the probability of positive reinforcement, and therefore to approach the attractant, even though it receives information about the attractant's location coded in terms of the landmark odours. Once it has learned this, if the creature is placed in a new position within the environment, it heads directly for the attractant.

Barto also describes a multi-layered ANS, with topology as shown in Figure 3.2 [Bar86]. It learns to produce output identical to its input patterns, using only a global reinforcement signal, but takes some 15,000-20,000 training steps. This slow learning is due to the fact that some output nodes will be correct when $r=-1$ or incorrect when $r=+1$, and therefore many nodes may receive reinforcement which is actually misinformation. Barto and Sutton have worked on complex extensions of

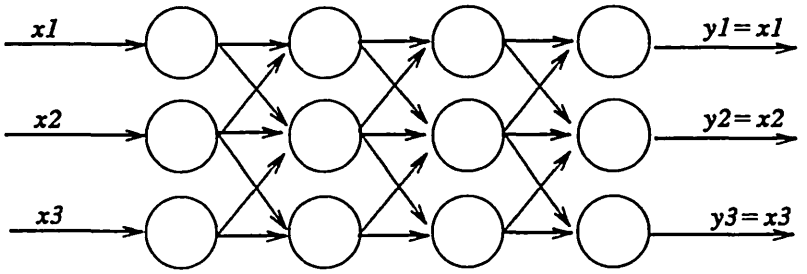


Figure 3.2. A multi-layered network of A_{R-P} units, learning to reproduce its inputs [Bar86].

the A_{R-P} algorithm for delay learning problems, and discussion of these is postponed until Section 3.3.3.

3.2.2. Constructing the desired output pattern

One way around this difficulty is to have the system use the reinforcement signal to generate a "desired" response, which the system can then be trained to mimic. This is the approach taken by Yeung [Yeu88], who constructed a system consisting of two adaptive components. One, the Associative Learning Network (ALN) learns to map signals from the environment to probabilities p_{ji} that each node j will output a 1. A second adaptive network, the Non-associative Learning Network (NALN), receives the reinforcement signal $r \in \{0,1\}$, and the output of the ALN, and generates a "desired" probability vector: $p_{ji}^* = \Delta p_{ji} + p_{ji}$, where if $r=1$:

$$\begin{aligned} \Delta p_{ji} &= \beta_1(1-p_{ji}) \qquad \textit{if } j \textit{ was the action chosen} \\ \Delta p_{ji} &= -\beta_1 p_{ji} \qquad \textit{if } j \textit{ was not the action chosen} \end{aligned} \tag{3.6}$$

and where if $r=0$:

$$\begin{aligned} \Delta p_{ji} &= -\beta_2 p_{ji} \qquad \textit{if } j \textit{ was the action chosen} \\ \Delta p_{ji} &= \beta_2 \frac{1}{|X|-1} - p_{ji} \qquad \textit{if } j \textit{ was not the action chosen} \end{aligned} \tag{3.7}$$

$\beta_1 > 0$ and $\beta_2 > 0$ are learning constants, and $|X|$ is the cardinality of the set of possible environmental inputs. Given this vector of "desired" outputs p_{ji}^*, the ALN is then trained by backpropagation on the difference $p_{ji}^* - p_{ji}$.

Yeung calls this "Linear Reward-Penalty", in comparison with Barto and Sutton's Adaptive Reward-Penalty learning. The A_{R-P} has only two outputs; when more are needed and A_{R-P}s are combined into the ASN, each element learns independently according to a global reinforcement signal, and this was shown above to lead to very slow learning. Yeung's system can have many outputs and can learn more quickly because each node receives a specific error signal.

3.2.3. Evaluation of reinforcement learning techniques

All of the reinforcement learning systems discussed accomplish learning of specific responses with non-specific feedback from their environments. In some cases, this reinforcement is simply a binary signal indicating the presence or absence of success. The usual approach to learning under these conditions is to apply changes to the nodes proportional to this reinforcement, and hope that after many cycles, the correct actions will have been those which most often resulted in positive reinforcement, and will therefore be the most strongly learned. Even Yeung's system, which constructs a "desired" output pattern uses this strategy to construct that desired output. However, particularly in the case of noisy reinforcement (such as experienced by Widrow et. al's blackjack player), this can result in very long learning times.

Le Cun et. al. note that while reinforcement learning is inefficient due to the provision of low-quality feedback, it is also easier to implement than supervised learning techniques such as error backpropagation, since the feedback information is usually a measure of error at the output layer, rather than a computation of this error at each node, and it is broadcast to every node alike [LGH89].

Reinforcement learning may therefore be desirable from the point of view of implementation; it is essential if learning is to involve domains where the desired output patterns are simply not available (as in some control problems) or not

defined (as in some game playing).

3.3. P3: Delay Learning

Delay learning in the context of this thesis refers to a restricted form of credit assignment: the association of actions, undertaken in a particular environment, with their results or reinforcement; even if that reinforcement is delayed and even if reinforcement from other actions intervenes.

There are basically three ways in which delay learning has been approached in the ANN literature: by mapping to a pattern association task, by maintaining histories, and by use of successful prediction as a reinforcer. These are discussed next.

3.3.1. Mapping onto a pattern association task

Neural networks are inherently pattern associators: given a certain combination of inputs, they produce a combination of outputs. This process may be stochastic, iterative, recursive, etc. But every neural network learning task, from optimisation to pattern classification to pattern storage, may be reduced to an associative mapping.

Therefore, perhaps the simplest way in which to use ANNs in conditions of delayed reinforcement is to translate the problem to be learned into a straightforward pattern association task. That is, for every input, the system simply memorises the correct output. It does not need to be aware of the complex timing system by which reinforcement may be arriving.

The car-driving system of Shepanski and Macy discussed in Section 3.2.1 [ShM88] is of this sort. It learns an association problem: given a distance to the pace car, it should output the correct velocity change. When the ANN makes a decision, it is rewarded or punished immediately. This is a simplification from the problem apparently being learned: where reinforcement would be delayed until a series of

incorrect outputs would result in a crash into the car ahead.

A similar approach is used in the backgammon player built by Tesauro and Sejnowski [TeS88]. In a game environment, reinforcement ordinarily only arrives at the end of a game (winning-positive or losing-negative). The backgammon player, however, learns by being shown examples of a database of game positions, together with the best possible move for each. It learns to mimic the decisions of the database by error back-propagation, and eventually can generalise to the myriad game states not explicitly represented in the database.

Tolat and Widrow [ToW88] updated the inverted pendulum system of Widrow and Smith [WiS64] to include "visual" input. Instead of the four variables of pendulum angle, pendulum velocity, cart location and cart velocity, its input consists of two 5x11 pixel frames showing the current and immediately previous state (velocity can be deduced from the two static pictures). They provide reinforcement in the form of a skilled human's decisions about the correct output in each state, and the ADALINE simply has to learn to associate the input patterns with the correct output actions.

There are additionally several systems which teach robot arms to execute trajectories using this method (e.g., [KuR89], [MKS88], [deC86], etc. – discussed further in Section 3.1.1).

Systems of this type abound and are quite successful at their required tasks. But the truck-backer and robot arm controllers perform supervised learning: outputs are compared with the predefined desired outputs, and the network is then trained by error back-propagation. The backgammon player compares network output with a carefully constructed evaluation function. Mapping a delay learning problem onto a pattern association problem can be expected to work well where one of these two techniques are possible.

3.3.2 History maintinance

An obvious way in which to bridge the gap in delay learning between the output and the arrival of results is simply to keep a history or trace of the outputs generated at each step. When the results arrive, each decision can be "re-played", and rewarded or punished according to the results. This history can take the form of an explicit buffer, or it may be maintained locally and implicitly by an eligibility trace at each node.

3.3.2.1. History buffers

Several of the systems mentioned in previous sections have used history buffers to do delay learning. Cecconi and Parisi's food-finding creature (Section 3.2.2, [CeP89]) was trained by storing past input/output pairs. When food was encountered, the creature was moved back eight steps, and the decision made at each of those eight steps was rewarded. The truck backer (Section 3.2.1, [NgW89]) and the blackjack player (Section 3.3.1, [WGM73]) both save entire sequences until the truck hits the loading dock wall or the game is finished. Then each move in the sequence is rewarded or punished depending on this final result. Widrow and Smith's inverted pendulum balancer (Section 3.2.2, [WiS64]) performs exploratory learning, but remembers its sequence of movements; when the external observer judges a string of moves worthy of (positive or negative) reinforcement, each of the moves in that string is recalled and reinforced.

There are also a few examples of this sort of learning which do not involve ANNs per se. One is Langley's rule-based system which learns to solve the towers of Hanoi task [Lan85]. In this task, there are three poles and a number n of disks of different radius. The disks start on one pole, stacked in decreasing size. The task is to move the disks one at a time so that they finish in the same order on a different pole, with the constraint that a disk may not be placed on top of another disk with

smaller radius. His system constructs rules which are generalised with success and specificated with errors, and notes complete solution paths (as well as loops, dead ends, and negative conditions), which may then be reinforced.

The use of a history buffer is successful in limited domains (such as games, one-dimensional movements, and the like). Its chief shortcoming is that it requires the buffer, which is of size n. If n is too low, the system will not be able to learn a problem which involves delays of $n+1$ or greater. However, as n grows, the space needed to store the buffer also grows. Worse, the number of actions reinforced each time a result arrives also grows; there will be unnecessary (and possibly harmful) reinforcement of states which occurred many steps ago and which are unrelated to the actions giving rise to the results.

3.3.2.2. Eligibility traces

These considerations have lead some workers to adopt eligibility traces rather than buffers to maintain histories. Each input and possibly output is given its own trace: which is a value that increases when the input (output) is active, and decreases to zero with time in the absence of further activity. Then, when reinforcement arrives, each synapse is adjusted as a function of its eligibility – those which were active most recently will be affected the most strongly. These traces do not constitute a buffer, because re-occurrence of the same input will overwrite an earlier trace. Space requirements are constant (and linear in the number of variables keeping traces); the longest possible trace depends not on the space used but on how long it will take for an eligibility to decay to zero.

The idea of using eligibility traces goes back at least as far as to Mitchie and Chambers's BOXES system (Section 3.2.1, [MiC68]). In that system, each possible state maintained its own eligibility, which was increased each time that state was entered, and otherwise decayed to zero. On reinforcement the last move made in

each state was (positively or negatively) reinforced, proportional to that state's eligibility.

Witten's controller for Markov environments is similar [Wit77]. Each possible environment maintains a list of potential actions, and the expected eventual reinforcement obtained if each action is selected (discounted by how long it will be until the expected reinforcement arrives). When reinforcement arrives, the expectation is updated proportional to the strength of the reinforcement as well as how long it took to materialise; this is simply the same as rewarding each state according to a decaying eligibility.

Examples of this approach are found in the work of Barto and Sutton on modelling classical conditioning (CC). In CC, an unconditioned stimulus (UCS) exists which when applied elicits a response R. During training, a conditioned stimulus CR is repeatedly presented just before UCS. Eventually, the CR alone will evoke the R: it has become a predictor of the UCS, and hence of R. For example, a dog's innate response R to the sight of food (the UCS) is salivation. If a tone repeatedly precedes the UCS, the tone becomes a CS and the dog will salivate in response to the tone, even if no food is then given.

CC involves delay learning when the CR precedes the UCS, which may be thought of as the reinforcer r in an ANN context, by some time interval.

Barto and Sutton use equations like the following (e.g., [SuB81]):

$$y_j(t)=\sum_{i=1}^{I} w_{ji}x_{ji} \tag{3.8}$$

$$\bar{x}_{ji}(t)=\alpha\bar{x}_{ji}(t-1)+x_{ji}(t-1) \tag{3.9}$$

$$\bar{y}_j(t)=\beta\bar{y}_j(t-1)+(1-\beta)y_j(t-1) \tag{3.10}$$

$$\Delta w_{ji}(t)=c\left[y_j(t-1)-\bar{y}_j(t-1)\right]\bar{x}_{ji}(t-1) \tag{3.11}$$

$\bar{x}$ and $\bar{y}$ are traces of the input and output activities respectively. When an input x_{ji} is active, it is "tagged" for some time after with a decaying eligibility $\bar{x}_{ji}$. If $\bar{y}_j(t)=y_j(t-1)$, then $y_j(t)-\bar{y}_j(t)$ is non-zero whenever node output changes. The

result is that if y_j changes while $\bar{x}_{ji}$ is non-zero, then the weight relating them (w_{ji}) changes – more depending on how recently x_{ji} was active.

Then the reinforcement to the system may be defined as:

$$r_{ji}(t) = \left[y_j(t) - \bar{y}_j(t)\right]\bar{x}_{ji}(t) \qquad 3.12$$

This provides negative feedback in the form of $\bar{y}_j(t)$, which is a weighted average of j's past output [SuB81].

This becomes a classical conditioning model when the input x_{j0} in this model is defined as the UCS, and it has a fixed efficacy w_{j0} which is sufficient to produce the CR y_j. The development of the conditioned response R to a CS is equivalent to production of y_j when some x_{ji} is present ($i \neq 0$). Such learning is shown in Figure 3.3.

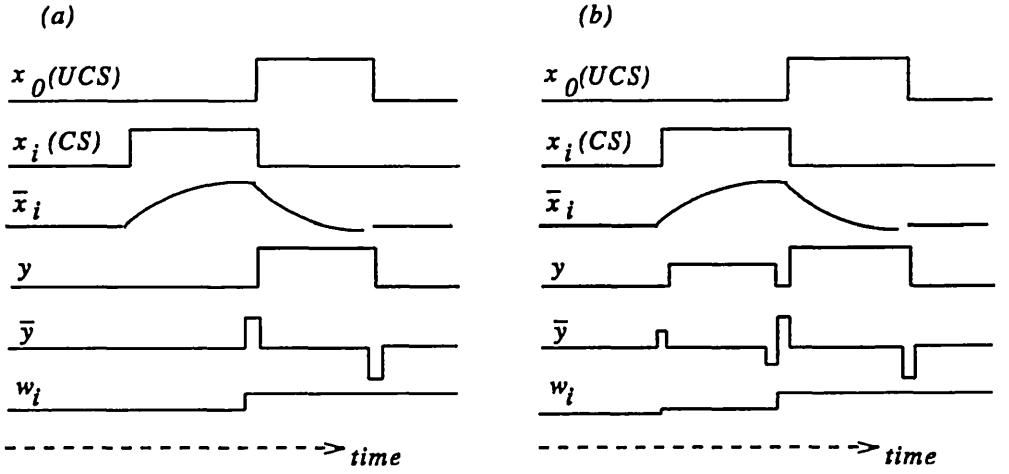


Figure 3.3. The responses of Barto and Sutton's system on the first pairing of CS and UCS (a), and on a later pairing (b) – showing the development of a conditioned response in a single node j. The weight w_i changes each time $y - \bar{y} > 0$; but when $y - \bar{y} < 0$ these changes are not undone because the eligibility $\bar{x}_i$ has decayed to zero.

Using this model, Barto and Sutton can also generate blocking, selective learning of most reliable or earliest predictors of the UCS, conditioned inhibition, and higher-order chaining of associations [BaS82].

Klopf's model of conditioning is related to that of Barto and Sutton – in essence, it is a compromise between maintaining buffers and eligibility traces, in that

it saves the last few input/output pairs, and assigns a decaying eligibility to each. His weight-adjustment rule is [Klo86]:

$$\Delta w_{ji}(t) = \sum_{z=Tmin}^{Tmax} \alpha_z |w_{ji}(t-z)| \Big[x_{ji}(t-z) - x_{ji}(t-z-1) \Big] \Big[y_j(t) - y_j(t-1) \Big] \qquad 3.13$$

where x_{ji} and y_j are input and output frequencies and where $Tmax$ is the longest inter-stimulus interval (ISI)[1] over which conditioning is known to be effective (perhaps 3 seconds, in .5 second intervals [Klo88]); α_z is the learning rate constant, proportional to the efficacy of conditioning when ISI$=z$ ($\alpha_z > \alpha_{z-1}$). Only positive changes to x_{ji} are considered.

Weights are considered to be bounded so that they never cross 0; if it may be necessary for an input to be represented by both excitatory and inhibitory weights, two synapses are provided for that input – one of each kind.

System output is given by:

$$y_j(t) = \sum_i w_{ji}(t) x_{ji}(t) - \theta_j, \qquad 3.14$$

where θ_j is the node threshold.

Klopf [Klo86] compares this model with Hebbian learning and also with the Barto and Sutton model, and shows how it more closely models observed classical conditioning data in delay conditioning with various timings of CS duration. Also, while the change to a weight tends to be linear in Hebbian learning, and negatively accelerated in the Barto and Sutton model, Klopf's weights show a learning curve which is first positively accelerated and later negatively accelerated, which is also more like the biological situation.

The model reproduces a range of classical conditioning effects involving length and intensity of conditioned and unconditioned stimuli, and their timing with respect to one another [Klo88]. A delay between the CS offset and UCS onset results in

[1] The ISI refers to the time between the start of the CS and the start of the UCS. Effectiveness of conditioning tends to decrease as ISI increases, and is at a maximum when UCS follows the CS very closely in time.

slower and weaker learning in the model, as in animals, since the UCS is not only paired with the CS going on but also with it going off.

Like the Barto and Sutton model, the node itself supervises its own learning by detecting changes in the environment. No specific external reinforcement is needed. In this case, the UCS pathway provides a sort of external reinforcement: when the UCS is present, the node is forced to fire by the strong weight on that input line, and this is then the objective event with which the predictions (as evidenced by recent output) can be compared.

The major difference with Barto and Sutton is that explicit traces are kept in the $x_i(t-z)$, whereas Barto and Sutton keep only an average, $\overline{x_i}(t)$. This allows Klopf's model to succeed on delay conditioning tasks with timing eccentricities such as CS going off before the UCS comes on, where the Barto and Sutton model would fail. By doing so, Klopf's model manages delay learning as a matter of course, whereas Barto and Sutton enable their model to do delay learning only by providing the extra mechanism of prediction-driven learning (Section 3.3.3).

McAuley used Klopf's model to implement a system which learns to track moving objects across a two-dimensional screen [McA88].

Taylor and Gorse have also used a learning rule including eligibility traces to train i-pRAMs [GoT90]. The rule includes eligibility traces which represent correlation between activity at address X in node j and activity (e_{Xj}) or inactivity (f_{Xj}) at the node output:

$$\Delta s_X^j = \alpha \left[(y_j - s_X^j) r + \beta (1 - y_j - s_X^j)(1-r) \right] \left[e_{Xj} y_j - f_{Xj}(1-y_j) \right] \qquad 3.15$$

where e_{Xj} and f_{Xj} replace the x_i in Equation 3.4:

$$\begin{aligned} e_{Xj}(t+1) &= \gamma e_{Xj}(t) + (1-\gamma) y_j X(t) \\ f_{Xj}(t+1) &= \gamma f_{Xj}(t) + (1-\gamma)(1-y_j) X(t) \end{aligned} \qquad 3.16$$

As usual, s_X^j is the value stored in pRAM j at address X.

This is basically the same as the other rules except that it is adapted to train stored values rather than weights, and the positive and negative reinforcements as well as the active and inactive outputs are treated separately.

Gorse and Taylor have applied a system of 4 i-pRAMs using this learning rule implement a food-finding creature operating in the world specified by Barto and Sutton (Section 3.2.1, [BaS81]). It learns to move efficiently to the attractant, given only information about the distance of the landmarks, within some 300 training cycles [GoT90].

3.3.3. Prediction-driven learning

Eligibility traces and history buffers, such as those used in the sections above, work well in domains where the reinforcement is frequent (so that buffers are small or only a few states are eligible) and where there is a balance between positive and negative reinforcement.

Unfortunately, many interesting problems are not that simple. For example, the inverted pendulum problem described in Section 3.1.1 has been dramatically simplified by some of the systems so far which claim to solve it: it has been mapped onto a non-temporal problem [SaS89], or else a human observer has been allowed to reinforce strings of moves [WiS64]. The BOXES system tackles the full problem [MiC68]: reinforcement only comes at the end of a potentially very long sequence, and it is only ever negative in sign (indicating that the pendulum has fallen over or that the cart has overrun its track). The penalty for this completeness is that learning in BOXES is very slow, taking at least several hundred trials before balancing is maintained for any length of time [BSA83].

An alternative is to reinforce the system on the basis of a feedback signal which is available at every single cycle, even if proper external reinforcement is not. Such a feedback signal can be defined as the system's "prediction" or "expectation"

of future reinforcement.

The technique dates back to Samuel's checkers player [Sam63]. That system executed a move, and the resulting new board state was evaluated. This new evaluation was then used to modify the evaluation of the previous board – i.e., to update the expected game outcome from that point. A sequence of moves is rewarded if its immediate or near-immediate successors have a positive evaluation, and punished if they do not. At each move, the current board score is compared with that from before the move, and the difference between the two is noted. If the score has changed, the earlier score may be corrected: "we are attempting to make the score, calculated for the current board position, look like that calculated for the terminal board position of the chain of moves which most probably will occur during actual play" [Sam63].

Barto et. al. have proposed a rule which calculates reinforcement based on $r(t)-p(t-1)$, where $p(t-1)$ is the system's prediction of $r(t)$ [BSB81]:

$$\Delta w_{ji}=c\Big[r(t)-p(t-1)\Big]\Big[y_j(t-1)-y_j(t-2)\Big]\bar{x}_{ji}(t-1) \qquad 3.17$$

They then use a separate predictor node, as shown in Figure 3.4, which learns to output $p(t)$ and provides that prediction to all of the other nodes. Then, weights are changed not when a change in r co-occurs with a change in y_j, but when a prediction p is proven wrong. The network shown in Figure 3.4 is termed an Associative Search Network [BSB81], and the learning method is often termed Adaptive Heuristic Critic (AHC) learning [Sut88].

A famous instantiation of this is the inverted pendulum balancer [BSA83], which consists of two nodes, the predictor – the Associative Critic Element (ACE) – and a single node to learn the proper output – the Associative Search Element (ASE). The problem is simplified in that the environmental information is passed through an encoder, so that each possible input state activates a unique ASE and ACE input, and thereby has its own associated weight.

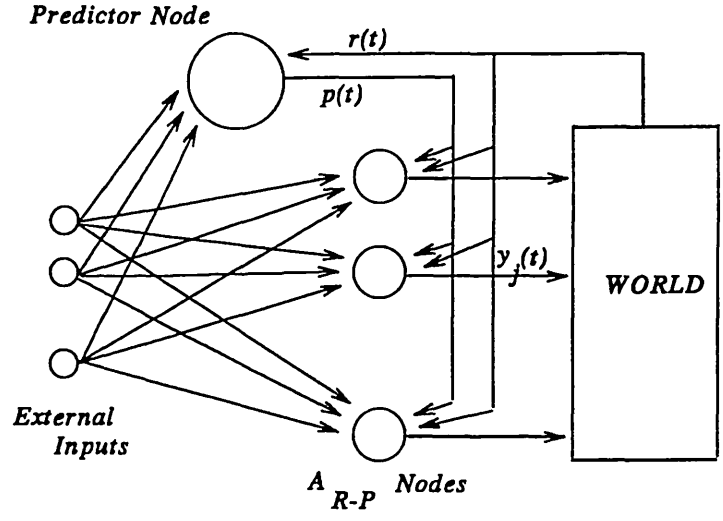


Figure 3.4. The Associative Search Network [BSB81]. The predictor node (P) learns to output $p(t)$, a prediction of $r(t+1)$, and the remaining nodes use this to update their weights according to Equation 3.17.

The ACE learns mappings from input plus movement to predictions of future reinforcement. ACE weights are updated according to:

$$\Delta w_{ACEi} = \beta \left[r(t) + \gamma y_{ACE}(t) - y_{ACE}(t-1) \right] \bar{x}_i(t) \qquad 3.18$$

where β and γ are learning constants. $r(t)$ is the reinforcement signal, which is always equal to 0 except when the pendulum falls or the cart crashes, whereupon $r(t) = -1$. $\bar{x}_i$ is the usual input trace. The ACE compares current with previous predictions, and updates its weights when these predictions are in error.

Meanwhile, $\hat{r}(t)$, the reinforcement signal to the ASE, is computed as:

$$\hat{r}(t) = r(t) + \alpha y_{ACE}(t) - y_{ACE}(t-1) \qquad 3.19$$

The ASE learns mappings from the input space to $y_{ASE} \in \{+1, -1\}$, representing right and left moves of the cart; and ASE weights are changed as:

$$\Delta w_{ASEi} = \delta \hat{r}(t) \bar{x}_i(t) \qquad 3.20$$

Each input has an eligibility trace x_{ASEi}, which is set to the ASE output when that input x_i is active, and which decays to 0 thereafter.

ASE weights are negatively reinforced when predictions of the ACE are

wrong, not when explicit failure occurs. If $r(t)=0$, then if the last move holds a better prediction of reinforcement than the current one (meaning $y_{ACE}(t-1)>y_{ACE}(t)$), then the ASE is punished for its last output; while if the current prediction is better (meaning $y_{ACE}(t-1)<y_{ACE}(t)$), then the ASE is rewarded. If $r(t)=-1$, then $\hat{r}(t)=-1-y_{ACE}(t-1)$: if the failure was fully predicted then no punishment occurs; otherwise recent actions are punished proportionately to their eligibility $\bar{x}_i(t)$.

Barto et. al. report that within 100 trials, the ASE/ACE system usually manages to balance the pendulum upright for more than 500 000 time steps, equivalent to nearly 3 hours of simulated real time [BSA83].

Rosen et. al. showed that by the addition of some domain-specific heuristics (namely, encouraging moves which cause the system to traverse a cycle through state space, and encouraging short cycles more than long ones), the ASE/ACE system can be made to learn much faster than that – within 100 trials. [RGV88]

ASE/ACE systems have been applied with success to problems such as maintaining a submarine at a constant distance from an irregular ocean floor [PoC90], finding a set of actions to solve the towers of Hanoi problem [Che90], and implementing an adaptive setpoint controller [GuM90].

The idea of prediction-driven learning is extensible to the idea of learning by Temporal Differences (TD) [Sut88]. In this case, learning is not based on the difference between the predicted and actual outcome, but on the difference between the current and last predictions themselves. Weight change is based on a rule such as:

$$\Delta w_{ji}(t)=\alpha(p_j(t+1)-p_j(t))\sum_{k=1}^{t}\lambda_{t-k}\nabla w_{ji}p_j(k) \qquad 3.21$$

Here, $p_j(t)$ is the predicted output of node j at time t. $\nabla w_{ji}p_j(t)$ is the partial derivative of the prediction at time t with respect to a change in weight w_{ji}, and the λ_{t-k} are learning constants.

Sutton justifies this approach on the grounds that there are several situations which this system could handle which an AHC system could not [Sut88]. One example is a world where a state N is usually followed by negative reinforcement with probability of 90%. A novel state A is entered, which leads to N, and on this instance N is followed by positive reinforcement. The AHC will (incorrectly) positively reinforce the move from A to N. The TD system will (correctly) assign some negative reinforcement to A based on the *expected* reinforcement following state N. On the other hand if the transition from A to N is followed by negative reinforcement, the AHC system will assign a 100% expectation of negative reinforcement to state A; whereas the TD system will, correctly, assign merely a high likelihood of negative reinforcement to state A.

Sutton also lists some situations where TD will fail [Sut88]. If there is a situation where state N entered from A brings positive reinforcement, but the entry of N from any other state leads to negative reinforcement, the TD methods will always assign expected reinforcement to A based on the usual expected negative reinforcement from state N.

Several other workers have constructed systems using variants of AHC and TD methods.

Williams's REINFORCE algorithm implements weight changes proportional to changes in reward and eligibility of the weight [WiP89]; he notes that this and the AHC can both be generalised to a canonical form of delay reinforcement learning [WiP89]:

$$\Delta w_{ji}(t) = \alpha p(t) \bar{x}_{ji}(t) - \delta w_{ji}(t) \qquad 3.22$$

where $p(t) = r(t) - \bar{r}(t-1) + \beta$, and there is a trace of the reinforcement $\bar{r}(t) = \gamma \bar{r}(t-1) + (1-\gamma) r(t)$.

Schmidhuber has extended AHC learning to deal with a recurrent network, in place of a single-layer or feedforward action-outputting network [Sch90a]. Like in

AHC, a critic network learns a mapping from the state of the action network to prediction p of reinforcement. It is trained by EBP on the error $p(t)-r(t)$, if reinforcement arrives at time t, else simply on $p(t)-p(t+1)$. The recurrent action network then learns its mapping from external input to node outputs y_j on the basis of the learning rule:

$$\Delta w_{ji}(t)=\lambda\epsilon(t)y_i(t-1)y_j(t) \qquad 3.23$$

The system is shown to do delayed exclusive-or detection and the inverted pendulum problem [Sch90b].

Werbos has worked with systems in which the critic network is trained (by supervised learning) separately and first, which speeds later learning in the action-producing network [Wer90].

The idea of learning by comparing changes in predictions also has some support in psychological data; Hull in particular suggested the idea of reverse chaining, whereby an animal learns first about states which lead to reinforcement, and then about earlier states which lead to those states, and so on [Hul32]. Through a phenomenon known as secondary conditioning, states which are known to lead to reinforcement can themselves becomes inherently reinforcing. Fulcher has constructed a system which does reverse chaining to accomplish delay learning [Ful89]. The system consists of a action-generating network which learns mappings from input states to moves, and a second network which learns to recognise secondary reinforcers – by mapping from input states to a prediction of reinforcement. When this second net recognises a reinforcing state has been entered, it causes a reward to be given to the last move of the action-generating network. The similarities with AHC are obvious: in both, one net learns to output the correct actions, while a second module learns to predict external reinforcement, and provides internal reinforcement to the action-generating network. Fulcher's system was shown to be able to solve simple maze traversal tasks. By making reinforcement values decrease with

distance in time from the primary reinforcer, the system could find the shortest path through the maze or handle multiple goals.

Finally, it is worth noting that there are systems which do not involve neural networks but which use prediction-driven reinforcement. An example is Booker's rule-based food-seeking creature, which uses verification or falsification of its predictions as a feedback signal [Boo88]. The system operates by trying to improve its predictions, rather than by explicitly trying to find food; motivators such as hunger as used however to determine the relative effectiveness of stimuli in eliciting behaviour.

3.4. Summary of Chapter 3 and Conclusions

The two basic means of satisfying P1 are learning by trial and error and learning by doing. It was shown that the former is thinly-disguised supervised learning, while the latter generally involves reinforcement learning.

Reinforcement learning strategies include the use of a global reinforcement signal (or use such a signal to construct a "desired" output). Their principal shortcoming is very long training times, compared with supervised learning systems, because the feedback information is less rich.

The MPLN learning algorithm presented in Section 2.5 is a reinforcement learning algorithm which learns by doing, satisfying both P1 and P2. It was noted in Chapter 2 that, due to its RAM-based nature, learning is also quite fast. For this reason, MPLNs have been selected as the vehicle for the current study.

Delay learning may be accomplished by mapping onto a pattern association task – but only for a limited or simplified problem class. It may be done by history maintenance using buffers or eligibility traces, or by use of prediction-driven reinforcement. Use of buffers of size n renders the system unable to learn over delays of length $n+1$; but as n grows so do space requirements and the probability of

reinforcing an irrelevant state from the distant past. Use of an eligibility trace requires a fixed amount of space but if the same state is entered twice before reinforcement arrives, the two events will be confounded. Worse, systems using simple eligibility traces do not work very well if delays are long or if reinforcement patterns are skewed. Such a problem arises in Barto and Sutton's inverted pendulum balancer [BSA83]. For this reason, they and others, have turned to prediction-driven reinforcement: where nodes learn according to signals assessing the (possibly long-term) consequences of actions. These systems are more powerful and quicker to learn. However, there are some basic tasks which they cannot master, such as the following:

$$\begin{aligned} &\textit{action A followed by action B} \rightarrow \textit{positive reinforcement} \\ &\textit{BUT action C followed by action B} \rightarrow \textit{negative reinforcement} \end{aligned} \qquad 3.24$$

The type of problem posed by Equation 3.24 is of interest to this thesis. Section 1.2 defines P3 to include situations where reinforcement may be delayed and where other outputs and reinforcements may arrive in the meantime: i.e., intervening states and reinforcements may have to be ignored. Yet neither a history-maintaining approach nor a prediction-driven approach can solve such a problem.

Accordingly, the system presented in the next chapter represents a compromise between the history-keeping and prediction-driven methods. It maintains a history buffer, but the states saved in this buffer are selected as a function of the network's predictions about their (possibly long-term) consequences.

It will be shown that in such a system, the buffer size n can be small and yet reinforcement can be delayed by more than n time units and the system can still learn. The system uses predictions to guide which states receive this reinforcement, but it can solve problems such as that in Equation 3.24 which prediction-driven (and simple history-maintaining) systems cannot.

CHAPTER 4: A MODEL TO SATISFY P3 – ATTENTION-DRIVEN BUFFERING

4.1. Introduction

In the first chapter of this thesis, three properties were required of the system under construction:

P1. That it be an exploratory automaton, capable of learning without a training phase.

P2. That it learn according to a global error signal or evaluation of its behaviour, rather than requiring provision of a desired optimal behaviour pattern which it then would try to mimic.

P3. That it be able to learn even when that global error signal is not immediate, and when other results may arrive in the intervening time.

Chapter 2 introduced the probabilistic logic node (PLN), and developed its extension, the multi-valued PLN (MPLN). It was claimed that an MPLN network could be constructed which exhibited properties P1 and P2. It remains to design an MPLN-based system which satisfies P3.

In chapter 3, several approaches were discussed relevant to this issue.

Use of an history trace (Section 3.3.2) is suitably general-purpose, but may require a great deal of memory to store past states. In cases where the history is a list of some n previous states, this paradigm disallows the system learning an association of order $n+1$. The advantages of this approach are that it requires little *apriori* design knowledge, and that it has been successfully used on problems illustrating P3. As such, this method is drawn upon in the model described later in this chapter.

A second strategy, prediction-driven learning (Section 3.3.3), is also a primary influence on the model. In this approach, the net makes predictions about future

world states (as reflected in the net's subsequent inputs). Positive reinforcement occurs when this prediction proves accurate; negative reinforcement comes when the predictions differ from what is experienced. These nets learn by minimising surprise rather than by maximising some external reward – and may therefore be described as curiosity-driven models. However, learned response to a state is inextricably linked with the responses to the states which precede or succeed it in time. Yet one of the conditions of P3 is that the system should be able to learn a response independently of intervening events and reinforcements.

The system proposed here, attention-driven buffering (ADB), does keep an explicit history – a buffer – of past states. Because of space constraints, this history is constrained to be of finite size: only a few previous states may be buffered at any one time. The machines discussed in Section 3.3.2, buffering only the n immediately previous states, are restricted to learn associations of order n or less: if the results of an action occur $n+1$ time steps later, they will never be paired with the correct state.

For this reason, in ADB, the retained states are not necessarily the n immediately previous ones. Rather, they are those involving the n most unfamiliar recent stimuli. A state is placed into the buffer when its outcome is more unpredictable than that of some state currently stored, and it will then oust that state and take its place. Simultaneously, the longer a state has been in the buffer, the more likely it is to be ousted by a new state. If the effects are predictable with great certainty, it is unnecessary that the new state enter the buffer – in fact, its entry might be harmful in that it might oust an state whose effects need to be learned.

This buffering on the basis of expectation (attention) rather than on the basis of chronological ordering is the cue which ADB takes from the concept of prediction-as-reinforcement. The predictions themselves are not explicitly reinforced, as in prediction-driven learning, but rather indirectly determine whether a

behaviour receives reinforcement by determining whether a pattern is still in the buffer when the associated results arrive some time after – even if the delay in arrival is longer than the size of the buffer.

In the remainder of this chapter, the ADB model is fully explained, and a simple example is used to show that the system can learn associations when the delay between response and reinforcement is longer than the size of the buffer – and thus a simple history trace would be insufficient. Attention is paid to how an ADB system might be implemented in hardware rather than in software simulation. Some analysis of parameters influencing the behaviour of the system is undertaken, and a series of machines implementing food-finding creatures show that the system actually works on larger problems.

4.2 The ADB Model

Define an *automaton* as a 5-tuple $\{J,F,Q,T,G\}$ [LeM89], where

$J = \{0,1\}^B$ is the input set
F = the finite set of states of the automaton
Q = the finite set of output actions of the automaton
$T : F \times J \rightarrow F$ = the state transition map
$G : F \rightarrow Q$ = the output map

Then the environment in which an exploratory learning system operates is an automaton itself.

The learning system is describable as a *learning automaton*, which is a 6-tuple $\{J,F,Q,T,G,Z\}$, where Z is a learning rule to update probabilities of production of various elements of α [LeM89]. It is also convenient to think of the input set of a learning automaton as $J=\{0,1\}^B \bigcup R$, where R is the set of possible reinforcement signals included in the output of the environment automaton and the input of the learning automaton. The interaction of an environment automaton with a learning automaton is often as shown in Figure 4.1.

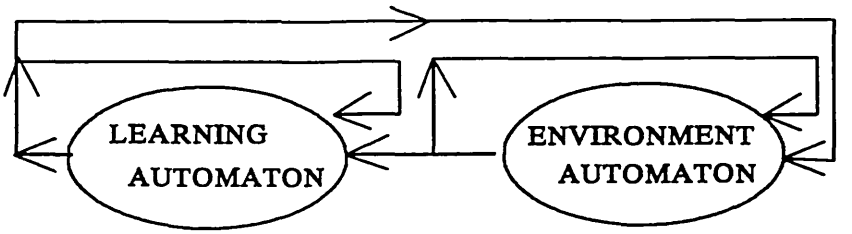


Figure 4.1. The interaction of an environment automaton with a learning automaton.

If the learning automaton is a reinforcement learning system, then R consists of some subset of possible reinforcements $\{-1,0,+1\}$, and its learning rule T should adapt output probabilities so that positive reinforcement is maximised and negative reinforcement is minimised.

Figure 4.2 shows the learning automaton considered in this chapter.

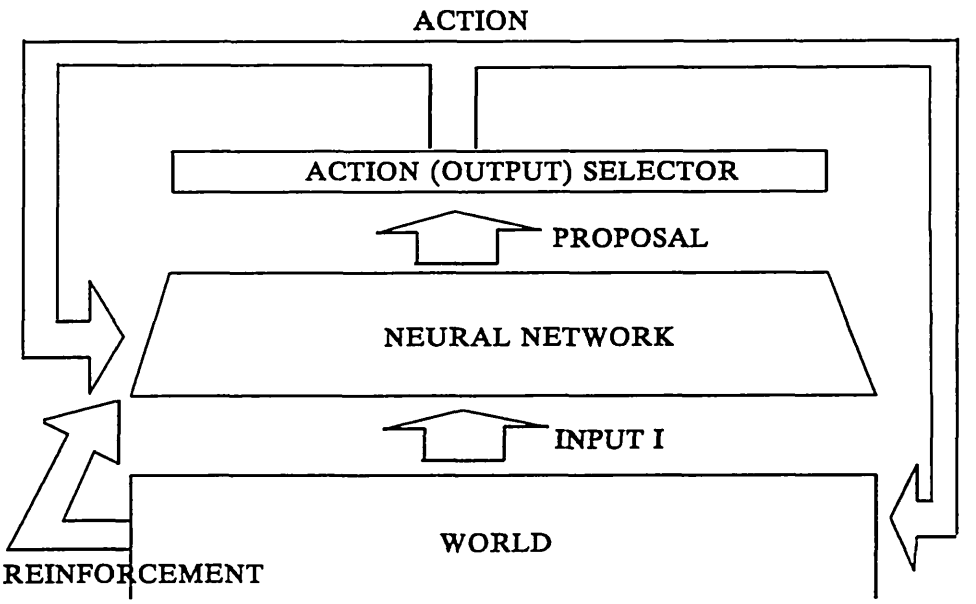


Figure 4.2. A learning automaton which generates actions based on proposals of an associative network.

Input from the environment automaton is converted into a binary-pixel input pattern, and this is passed to an associative memory which generates a proposed appropriate output for the learning automaton. This proposal is then passed to an output-generating subsystem, which translates this proposal into the actual learning

automaton output, by selecting from the automaton's output alphabet. There is also input from the environment automaton which serves as a three-valued reinforcement signal, indicating positive, negative, or no reinforcement. The learning automaton updates its output map, by adjusting probabilities of generating various proposals, on the basis of this reinforcement signal.

The system output is based on the proposal made by the learning automaton's associative memory. If the output alphabet is binary, then the outputs may be interpreted as "act" and "don't act" commands or as decisions to "accept" or "reject" the current input pattern. The proposal may be interpreted as a measure of the strength or sureness with which the associative memory predicts action is desirable in the current state. A strong proposal indicates an action is likely to result in positive reinforcement, a weak one indicates negative reinforcement is likely, and an intermediate one suggests no clear prediction is possible and the learning automaton output will have to be a guess.

To satisfy the demands of P3, delay learning, the learning automaton must be able to cope when the reinforcement signal is delayed and interleaved with other feedback. To accomplish this, a buffer is provided within the automaton to store past states until reinforcements arrive. Because the buffer is of bounded size, past states are not kept indiscriminately.

The entry of eligible states into the buffer is governed by an *attention* parameter, assigned to each state as it is entered. This attention is at a minimum when proposal strength is highest or lowest, and at a maximum when proposal strength is in the middle of its range. In effect, it indicates how "predictable" (positive or negative) reinforcement is in response to an action in the current state.[1]

[1] Use of the term attention to reference unpredictability or surprise follows from the work of Grossberg and Carpenter on Adaptive Resonance Theory (ART) [Gro80, CaG88]. ART deals with unsupervised learning – categorising inputs rather than associating them with external reinforcements. But it involves attention parameters. An input pattern generates a categorisation which then generates a prototypical pattern: in effect, it reports what it expects that the input looked like since it generated that categorisation. If the actual and prototype input patterns are sufficiently different, there is an attentional reset, and a different category is tried (or constructed). In ART, then, attention is used to refer to a condition

If the attention for the current state is higher than for at least one currently buffered element, that element is ousted and its place is taken by a new element representing the current state, and consisting of the input pattern, the current attention, and the proposal produced. If the associative memory is a neural network, its proposal may take the form of an output pattern. With each time step an element remains in the buffer, its attention decays toward zero.

When reinforcement arrives, each element in the buffer is accessed. If the attention is non-zero, the input pattern is reapplied to the associative network, and the decision to act is reinforced positively or negatively as the reinforcement signal indicates. This will encourage or discourage the automaton to repeat its previous response when next the input pattern appears.

Therefore, at any given time, the elements in the buffer represent the most recent, most unpredictable states entered. In this way, a compromise is made between history maintenance (which has high space requirements and limited trace length) and prediction-driven reinforcement (which cannot separate continuity in time from contingency). This form of learning is termed Attention-Driven Buffering (ADB), and is capable of associating input patterns with appropriate output actions when reinforcements occur with some delay, and if other reinforcements intervene; it is also able to learn across delays which are longer than the size of the buffer.

This is shown in a small example.

A system was posed the problem of learning to accept all of the patterns in Figure 4.3 except the last, which it should "reject" or fail to accept. The patterns were presented to the system as input in sequential order, so that the negative pattern appeared every eleventh time step. This would not be a particularly taxing problem

---

involving novel inputs to which the system is not sure how to respond, and it causes the current response to be suppressed and a new one tried. In ADB, attention is also used to refer to a condition involving inputs to which the system does not know how to respond, and it causes the current input and response to be buffered to await reinforcement information.

(the average Hamming distance between patterns is 0.430), but for the reinforcement schedule: following an acceptance, the results arrive three time steps after. The buffer used was of size $n=3$, and so without use of attention-driven buffering, three later states would have filled the buffer before the results arrived and could be associated with the appropriate input pattern.

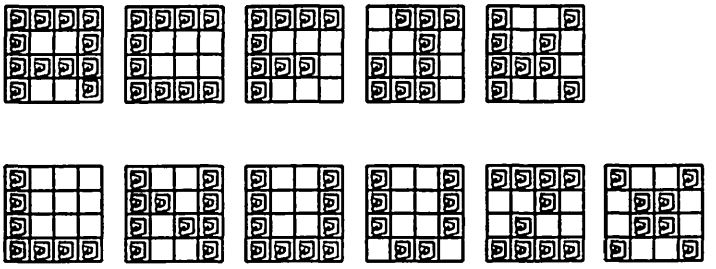


Figure 4.3. The training set for the example of Section 4.2.

The associative memory used contained twenty 4-input MPLNs as described in Chapter 2; each output "1" as a vote to accept or "0" to reject. The proposal was thus a rating from 0 to 20 for acceptance, and the output generator caused the system to output an accept decision if the proposal scored at least 18. During reinforcement, the buffered input patterns were reapplied, and addressed locations in the MPLNs were then adjusted by $\beta^+=0.05$ (positive reinforcement) or $\beta^-=-0.25$ (negative reinforcement). Negative reinforcement was weighted more heavily because the negative pattern only occurred once in every 11 pattern presentations. The system was considered to have learned successfully when 5,000 time steps elapsed without further negative reinforcement being encountered (or about 455 sweeps through the eleven-pattern training set); this 5,000 was not included in the calculations of learning time.

The results of training (averaged over 10 experiments) showed the system could, in fact, learn not to accept the negative element within an average of less than 4,000 time steps (std. dev. $>5{,}000$) or less than 400 sweeps through the training set.

Figure 4.4 shows that this data is heavily skewed: over half of the trials finished within less than 1,000 time steps, and only two trials took over 3,000 time steps.

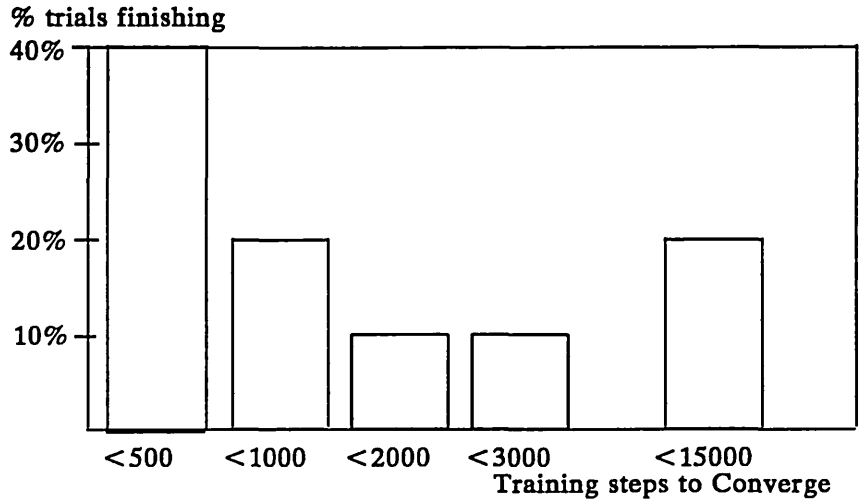


Figure 4.4. Results of training with delayed reinforcement on the data of Figure 4.3 by an MPLN-based ADB system.

An alternative measure of learning speed is the number of negative reinforcements encountered overall, and this averaged 21.3 (std. dev. 26.0).

Not all nets trained learned to accept all 10 positive elements, however. In particular, since the patterns always appeared in sequence, patterns A, C and F were always more recent than X when the negative reinforcement arrived. 60% of the nets trained did learn to accept all positive patterns, A, C and F included, and thus solved the problem perfectly; overall, the nets averaged a consistent acceptance of 9.1 of the 10 positive patterns – or an average of less than one falsely rejected.

This shows quite satisfactory learning of a reasonably difficult problem: learning with delayed reinforcement even when the delay length is larger than the buffer size.

4.3 Implementation Issues

David Marr [Mar82, p.24-27] divided the task of understanding an information-processing system into three levels:

- **Computational Theory:** What is the overall goal, independent of any implementational issues?
- **Representation and Algorithm:** How can the theory be implemented (in algorithmic terms)?
- **Implementation:** How can the algorithm be implemented physically?

Marr argued that all three levels were equally important; although thus far this thesis has concentrated on overall goal (P1, P2, P3) and algorithm (Chap. 2-4), it is worth making a departure at this stage to consider the possibility of physical implementation of an ADB system.

There are two strategies for implementing an ADB system such as that shown in Figure 4.2: external and internal buffering. There is no difference between the performance of these two kinds of implementation, and two automata differing only in the location of their buffers have equal learning ability.

Figure 4.5a shows a system with external buffering. Each input pattern passes through the associative memory, and is eventually assigned an attention if action is taken. If this attention is greater than some element in the external buffer, the current input and attention replace that element. On reinforcement, each input pattern stored in the buffer with non-zero attention is reapplied to the associative memory. Each node in the memory takes its input from this pattern and reinforces itself accordingly. The space required for an external buffer scales as $n(B+k)$, for buffer size n, a B-bit external input pattern, and an attention coded in k bits.

A system with internal buffering is shown in Figure 4.5b. Here, the associative memory nodes consist of (for example) I-input MPLN nodes each augmented with its own $n(I+k)$ buffer. Once attention is calculated, each node compares this new attention with those of the elements in its internal buffer; if the new attention is higher than some resident element, a new element is constructed of the current I node inputs and attention, and this overwrites the resident element. Reinforcement

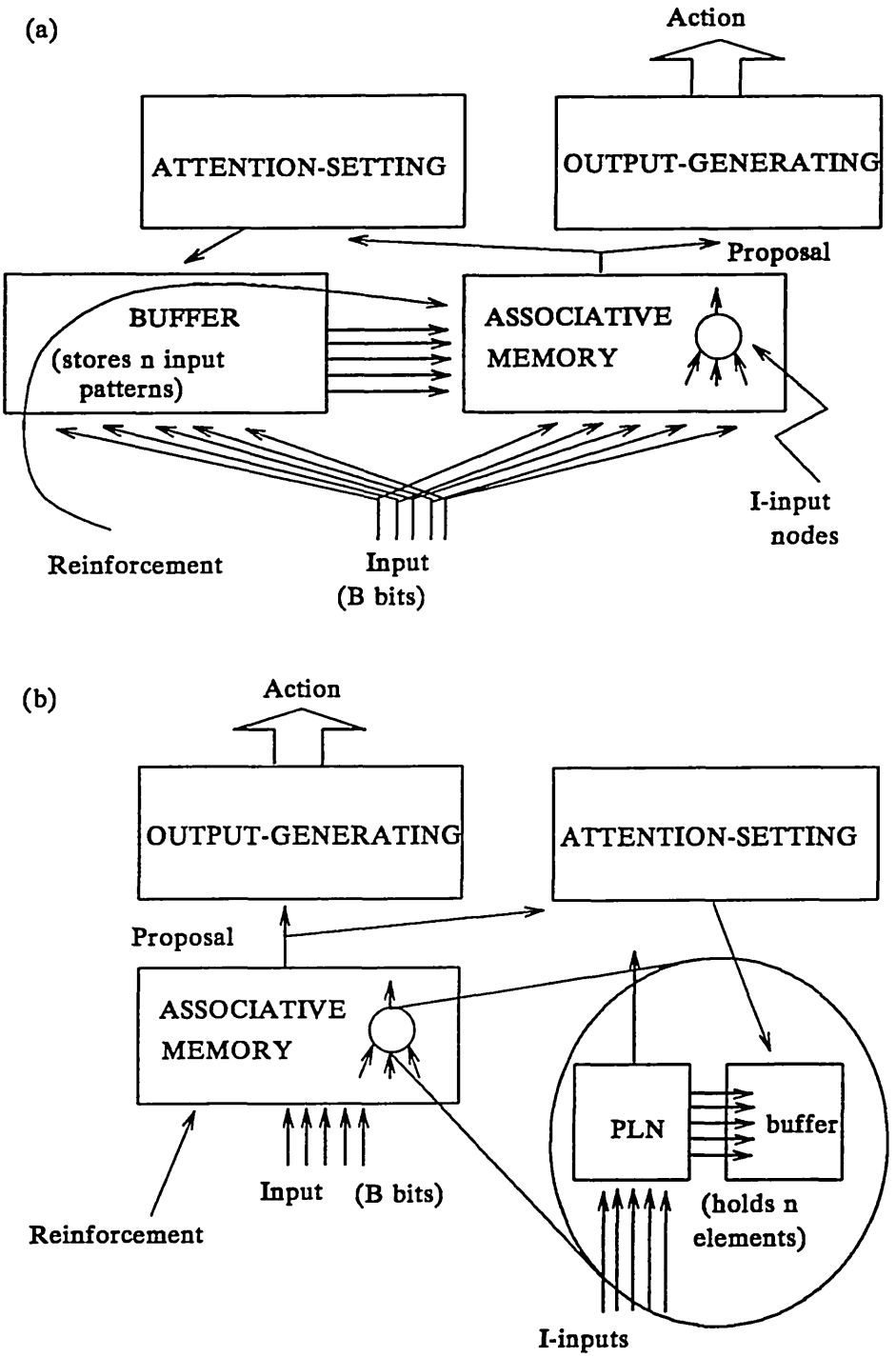


Figure 4.5. An ADB system with (a) external or (b) internal buffers.

causes each node to re-apply the I-bit buffered patterns (with non-zero attention) to its input lines, and update accordingly. Space requirements for an N-node net will be $Nn(I+k)$; for a single layer net, where $B \geq NI$, this will be considerably larger

than the requirements of the system with external buffering.

By maintaining an internal buffer at the node level (rather than externally at the system level) updating – putting elements into the buffer and also retrieving them and reinforcing the nodes – may be done locally and therefore the operation is massively parallel. If the updates are done on a system level, the operation in the best case will be constrained by the number of layers in the associative memory.

While the implementation would not necessarily be trivial, it is an encouraging feature of the ADB model that it can be implemented in this massively parallel, local, and distributed fashion.

In this thesis, mention will generally be made of "the buffer" to simplify discussion; such usage does not preclude the existence of many internal buffers at the implementation level.

4.4 Analysis of the ADB System: Delay and Buffer Size

The ADB systems considered in this thesis can be formally described as follows:

- Define an input regime which cycles through the training patterns $P_1, P_2, \cdots P_D$. If pattern P_1 is accepted by the system, reinforcement r occurs just before presentation of P_D. Then input P_1 is reinforced with delay D (and a delay $D=1$ implies immediate reinforcement).
- The time t is incremented following each sweep through the entire training set.
- Presentation of input pattern P_i elicits response $R_i(t)$ from the network, $0 \leq R_i(t) \leq 1$.
- Define the attention-generating function $f(R_i(t)) = [0,\alpha]$, where for all i:

$$f(R_i(0)) = \alpha$$
$$f(0) = 0$$
$$f(1) = 0$$

and where $f(x)$ is non-decreasing $0 \leq x \leq R_i(0)$ and non-increasing $R_i(0) \leq x \leq 1$ (f may however be a constant function).

- At time t, when input P_i is presented,

  if $i=1$, then $stm \leftarrow i$ and $attn \leftarrow f(R_i(t))$.
  else if $f(R_i(t)) > attn$, then $stm \leftarrow i$ and $attn \leftarrow f(R_i(t))$.
  else if $attn > 0$, then $attn \leftarrow attn - \delta$, $0 \leq \delta \leq 1$.

  The maximum attention α will then decay to 0 within $m = \left\lceil \frac{\alpha}{\delta} \right\rceil$ time steps.

- When r arrives, if $attn > 0$,

$$R_{stm}(t+1) \leftarrow R_{stm}(t) + \beta \quad (0 \leq R_{stm}(t+1) \leq 1)$$
$$R_i(t+1) \leftarrow R_i(t), \text{ for } all\ i \neq stm$$

  This is the simplest interesting ADB system, in that there is only one non-neutral input (P_1), and the buffer size is $n=1$.

First of all, it is easily shown that such a system can learn with delay D, even if $D > n$.

Result 1. *The maximum delay over which learning is possible is independent of buffer size, but dependent on the attention-assigning function f and on the memory increment δ.*

Proof. P_1 will be in the buffer ($stm = 1$) if

$$forall\ i,\ 1 < i < D,\ f(R_i(t)) < f(R_1(t)) - (i-1)\delta \qquad 4.1$$

and

$$f(R_1(t)) - (D-1)\delta > 0 \qquad 4.2$$

For $R_1(t)$ ever to be changed from $R_1(0)$ (and hence for any learning about P_1 to take place), Equation 4.2 must hold at least for $t=0$. Similarly, Equation 4.1 must hold for every i up to $i = D-1$, at least when for all $1 < i < D$, $f(R_i(t)) = 0$. Then these conditions may be rewritten as:

$$f(R_1(0)) - (D-1)\delta > 0 \qquad 4.3$$

This defines the maximum D over which learning can occur in terms of f and δ,

with n as small as 1.

Example. Consider $f(R_i(t))=\sin\pi R_i(t)$, $R_i(0)=0.5$, and $\delta=0.1$. Then by Equation 4.3, $D<11$.

Thus, it is possible that in an ADB system with n as small as 1, indefinitely long delays may be bridged using suitably constructed attention-assigning function f (and memory increment δ).

This is a simpler demand than showing that $R_1(\infty)\to r$, i.e., that the response to pattern P_1 is not only changed, but converges to the optimal response. In fact, this convergence can only take place if D is more constrained.

Result 2. *The maximum delay over which response $R_1(t)\to 1$ (or to -1) is independent of buffer size, but depends on f and δ.*

Proof. Consider $r=+1$ (the argument is symmetrical if $r=-1$); and again assume that all $f(R_i(t))=0$, $1<i<D$. During training, response R_1 will be reinforced some m times before $f(R_1(t))$ falls so low that Equation 4.2 is no longer satisfied and no more changes to $R_1(t)$ occur. In order for $R_1(\infty)\to 1$, $R_1(\infty)=R_1(0)+m\beta\geq 1$, or $m=\left\lceil\frac{1-R_1(0)}{\beta}\right\rceil$. R_1 must then generate sufficient attention (f) after $m-1$ reinforcements to generate still one more:

$$f(R_1(0)+(m-1)\beta)-(D-1)\delta=f(R_1(0)+(m-1)\beta)-(D-1)\delta>0 \qquad 4.4$$

Example. Given the same system as above, $f(R_i(t))=\sin\pi R_i(t)$, $R_1(0)=0.5$, $\beta^+=0.1$ and $\delta=0.1$, then by Equation 4.4, $D\leq 4$.

If reinforcements occur to more than one element, and particularly if both positive and negative reinforcements occur, the situation is more complicated, and the maximum D under ideal conditions may not be obtainable in practice. In particular, if pattern P_i is in the buffer when a reinforcement arrives, its response will be affected, even if that reinforcement was actually elicited by response to a different

pattern. In the worst case, if P_i is associated with several negative reinforcements elicited by patterns close to it in time, R_i may drop sufficiently that the exploratory ADB system never again accepts P_i and the mistaken response will be unchangeable. Alternatively, the decreased R_i may lower $f(R_i)$ enough that, even if P_i is accepted, it will be unable to enter the buffer, and nothing further will be learned about that pattern. In the example of Section 4.2, for example, the negative pattern was always eventually consistently rejected, but the positive patterns which occurred close to it in time occasionally were rejected after convergence.

This is similar to the continuity/contingency separation problem encountered by a simple buffer-maintaining or eligibility-using system, as mentioned in Chapter 3. However, at least in an ADB system, it is possible that these conditions can be overcome – although this cannot be guaranteed.

The problem considered in Section 4.2 (data as in Figure 4.3) was applied to ADB systems with n ranging from 1 to 5.[2] Figure 4.6a shows that there is a trend of decreasing learning time with increasing n, as each reinforcement can affect more stored patterns at once. Figure 4.6b shows learning speed in terms of the total negative reinforcements required before convergence; this decreases with larger n, as the negative pattern is more likely to still be resident in the buffer after D cycles if the buffer has larger capacity. Figure 4.6c shows that with increased n, the likelihood of positive patterns becoming associated with negative results drops to zero. This is due to a larger buffer allowing each element to remain resident longer, and therefore each positive pattern in the buffer will be positively reinforced up to n times, if the $n-1$ patterns which follow it are also positive.

---

[2] The remaining system parameters were: N (number of MPLN nodes) = 20, I (MPLN fan-in) = 4, 19 MPLNs outputting 1 needed for a system "accept", MPLN stored values initialised to 0.8, $\beta^+ = 0.05$, $\beta^- = 0.25$, $D = 3$, $\delta = 1$, $\Phi^P(x) = x$, f a step function as shown in inset of Figure 4.6.

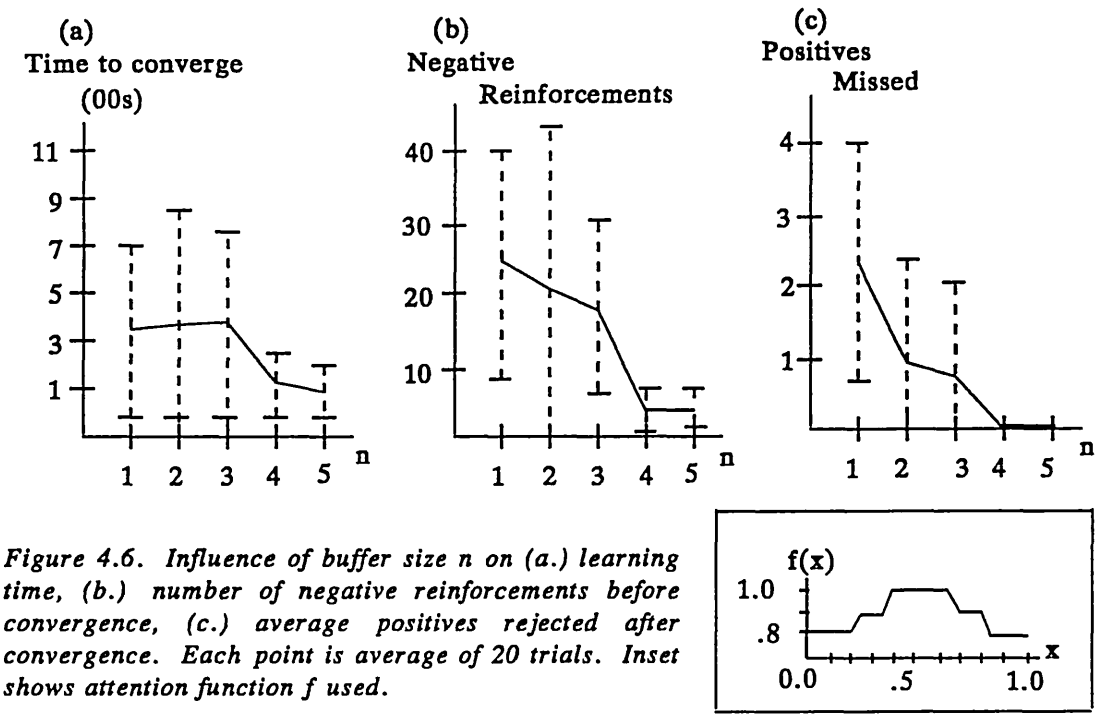


Figure 4.6. Influence of buffer size n on (a.) learning time, (b.) number of negative reinforcements before convergence, (c.) average positives rejected after convergence. Each point is average of 20 trials. Inset shows attention function f used.

4.5. Constructing MPLNs for the ADB System

In the examples above, values of $\beta^+ = 0.05$ and $\beta^- = -0.25$ were used to update the stored MPLN values.

A location addressed by a negative pattern receives average reinforcement denoted $\overline{\beta^-}$, which should be less than 0, to ensure that the node output tends toward zero for that input pattern. The longest a pattern can remain in the buffer is $\alpha' = \left\lceil \frac{\alpha}{\delta} \right\rceil$ cycles, during which the probability of a negative reinforcement arriving is $\frac{|M^-|}{|M|}$, and of a positive reinforcement is $\frac{|M^+|}{|M|}$. At the α'th cycle, if the element is still resident, it will receive its own (negative) reward. Therefore, the average reinforcement to a negative pattern should be:

$$\overline{\beta^-} = (\alpha' - 1)\frac{|M^+|}{|M|}\beta^+ + (\alpha' - 1)\frac{|M^-|}{|M|}\beta^- + \beta^- < 0 \qquad 4.5$$

For a positive pattern, the average reinforcement should be:

$$\overline{\beta^+} = (\alpha' - 1)\frac{|M^+|}{|M|}\beta^+ + (\alpha' - 1)\frac{|M^-|}{|M|}\beta^- + \beta^+ > 0 \qquad 4.6$$

In the above examples, $\alpha' = 5$, and there are $|M^+| = 10$ positive and $|M^-| = 1$ negative elements for a total $|M| = 11$. β^- was set to 0.25. Solving Equations 4.5 and 4.6 for β^+ yields $0.020 \leq \beta^+ \leq 0.09$; in fact $\beta^+ = 0.05$ was the value used.

This implies that the maximum number of possible values for an MPLN stored value is $\omega \approx 20$, which is higher than the value $\omega \approx 11$ suggested in Chapter 2. However, use of a smaller ω (and hence of a larger β^+) would have required a huge β^-, and so was not practicable.

Another MPLN parameter, Φ^P, in these experiments deviated from the value suggested in Chapter 2; it was $\Phi^P(x) = x$, rather than the steep sigmoid suggested theoretically in Section 2.6.2:

$$\begin{aligned} \Phi^P(sv) &= 0, \text{ if } sv < 0.5 \\ &= 1, \text{ if } sv > 0.5 \\ &= 0/1 \textit{ randomly}, \text{ if } sv = 0.5 \end{aligned} \qquad 4.7$$

Such an output function would mean that the response of a node to a given input is deterministic except in the case where the stored value is exactly equal to 0.5. Then, once a network has learned to reject all negative patterns, training effectively ends, except insofar as current acceptance of positive patterns will converge to probability of 1. If at this point, any positive patterns are not being accepted, there is no way for the network to recover from this state.

Alternatively, by providing the nodes with a softer, more probabilistic output function, it is possible that a pattern which has received negative reinforcement only once or twice, by virtue of appearing close in time to a negative pattern, will still be occasionally accepted. In this way, although learning may take some time longer to conclude than with a steep limiter, it is possible that the network can recover from failure to accept a positive pattern.

For this reason, the output functions used in most of the simulations which

appear in the remainder of this thesis are linear or approximations to linear functions.

A final MPLN parameter to consider is $\hat{s}$, to which stored values in the nodes are initialised at the beginning of learning. The probability of an untrained node outputting a one in response to a random input is $\Phi^P(\hat{s})$.

Because the machine being constructed is an exploratory automaton, it will never experience any reinforcement unless it outputs a decision to act. This argues that stored values in the MPLN nodes should be initialised to a value such that $\Phi^P(\hat{s})$ is close to 1. At time $t=0$, the probability that a random pattern will be accepted by the ADB system is:

$$Prob(acceptance) = \sum_{i=\psi}^{N} C_i^N(\Phi^P(\hat{s}))^i(1-(\Phi^P(\hat{s}))^{N-i} \qquad 4.8$$

where there are N nodes and at least ψ nodes must output "1" for the system to accept the input.

If, for example, $\Phi^P(x)=x$, $0.0 \leq x \leq 1.0$, then the probability of accepting a random pattern at time 0 is as shown in Figure 4.7, as a function of $\hat{s}$ and of ψ, with $N=25$. The figure shows that, particularly for high values of ψ, a high $\hat{s}$ is needed if the system is ever likely to generate any accepts.

For the case where Φ^P is a steep sigmoid, the situation is even more dramatic: for $\hat{s}<0.5$, the probability of acceptance is 0.0 for all ψ; for $\hat{s}>0.5$ it is 1.0 for all ψ. In this case, $\hat{s}$ must be greater than 0.5 for any accepts ever to occur.

However, using a high $\hat{s}$ means that more negative reinforcements will be required before the system learns to stop accepting negative patterns. After one negative reinforcement to a stored value, sv, in the average case:

$$sv^1 = \hat{s}+\overline{\beta^-} \qquad 4.9$$

while after m such reinforcements,

$$sv^m = \hat{s}+m[\overline{\beta^-}] \qquad 4.10$$

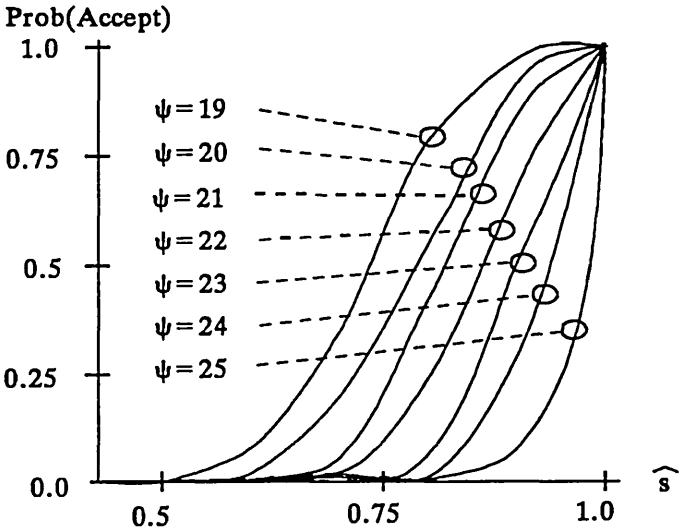


Figure 4.7. Probability an untrained system accepts a random pattern, as a function of $\hat{s}$, for several ψ. $N=25$; Φ^P linear.

Since convergence requires $sv^m \leq 0$,

$$m \leq -\frac{\hat{s}}{\beta^-} \qquad 4.11$$

Using Equation 4.5 and the values $\alpha=5$, $\frac{|P|}{|M|}=\frac{10}{11}$, $\frac{|L|}{|M|}=\frac{1}{11}$, $\beta^+=0.05$, $\beta^-=-0.25$, this becomes:

$$m \geq \frac{44}{7}\hat{s} \qquad 4.12$$

This is shown graphically in Figure 4.8. m grows linearly with $\hat{s}$: implying that higher values of $\hat{s}$ will result in more complete passes through the training set needed before the network is likely to converge. In simulations, $\hat{s}$ is often used as 0.8, as a compromise between the drives to increase $\hat{s}$ and to reduce it.

4.6. Construction of Network Topology

The remaining parameters in the ADB system are N, the number of nodes, I, the number of inputs to each node, and ψ, the number of votes needed for a system acceptance to occur. These are dependent on the problem to be learned: specifically

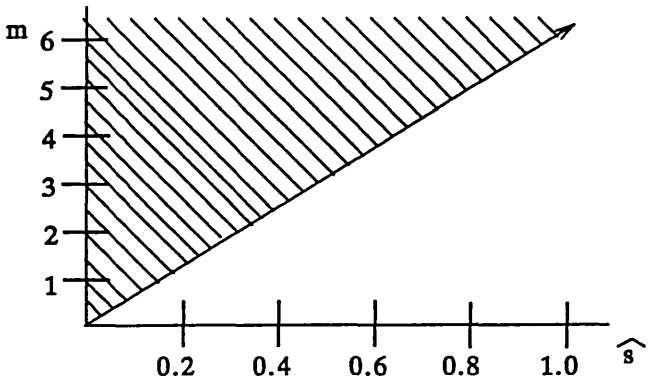


Figure 4.8. The relationship between m (indicating passes through the training set required before convergence) and ŝ. Shaded area: $m>6.28\hat{s}$.

on the percentage $\frac{|M^+|}{|M|}$ of positive patterns in the training set, or the probability of a positive reward in response to accepting an arbitrary pattern. As such, it is independent of the delay learning parameters such as D and n, and of the training algorithm, as characterised by β^+, β^- and δ.

Consider a training set, M, consisting of $M^- \subseteq M$ negative patterns and $M^+ \subseteq M$ positive patterns. To learn the problem, no more than ψ-1 MPLNs may vote for any $\nu \in M^-$, while at least ψ must vote for each $\mu \in M^+$. That is, each $\nu \in M^-$ will address one location in each MPLN, and at least $N-\psi+1$ of these must not be addressed by any $\mu \in M^+$; simultaneously, each $\mu \in M^+$ must address locations in at least ψ MPLNs which are not shared by any $\nu \in M^-$. These criteria are stricter than they would need to be in a multi-layer net because a single layer network is liable to oscillation of values if a location must be set two ways.

First, consider one $\nu \in M^-$ and one MPLN, j:

$$\begin{aligned} &Prob(\nu \text{ addresses the same location in } j \text{ as at least one } \mu \in M^+) \\ &\quad = \Lambda(\nu) \\ &\quad = \sum_{i=1}^{|P|} (-1)^{i+1} P(i,\nu) \end{aligned} \qquad 4.13$$

where $P(i,\nu)$ is as defined in Aleksander and Dobree-Wilson [AlD85], as the probability that an n-tuple of ν is sited in the overlap of exactly i elements of M^+:

$$P(i,\nu) = \sum \left(\prod_{k=1}^{i} A_{a_k,\nu} \right)^I forall \left\{ a_1, a_2, \cdots a_i | a_x \in M^+ \right\} \quad 4.14$$

In this equation, $A_{x,\nu}$ is the proportion of pixels x and ν share.

The probability that this occurs in at most $\psi - 1$ nodes, and hence that pattern ν is learnable is:

$$Prob(\nu \ learnable) = \sum_{x=0}^{\psi-1} C_x^N \Lambda(\nu)^x (1-\Lambda(\nu))^{N-x} \quad 4.15$$

The probability that all negative patterns $\nu \in M^-$ are learnable is:

$$Prob(M^- \ learnable) = \prod_{\nu=1}^{|N|} \sum_{x=0}^{\psi-1} C_x^N \Lambda(\nu)^x (1-\Lambda(\nu))^{N-x} \quad 4.16$$

The analysis for learnability of the positive training patterns is similar. First, consider one $\mu \in M^+$ and one MPLN, j:

$$\begin{aligned} &Prob(\mu \ addresses \ the \ same \ location \ in \ j \ as \ at \ least \ one \ \nu \in M^-) \\ &\quad = \Pi(\mu) \\ &\quad = \sum_{i=1}^{|L|} (-1)^{i+1} L(i,\mu) \end{aligned} \quad 4.17$$

where

$$L(i,\mu) = \sum \left(\prod_{k=1}^{i} H_{a_k}^{\mu} \right)^I forall \left\{ a_1, a_2, \cdots a_i | a_x \in M^- \right\} \quad 4.18$$

The probability that this does not occur is 1-($\Pi(\mu)$); the probability that there is no such overlap between μ and an element of M^- for at least ψ nodes, and hence that pattern μ is learnable, is given as:

$$Prob(\mu \ learnable) = \sum_{x=\psi}^{N} C_x^N (1-\Pi(\mu))^x \Pi(\mu)^{N-x} \quad 4.19$$

The probability that all $\mu \in M^+$ are learnable is therefore:

$$Prob(M^+ \ learnable) = \prod_{\mu=1}^{|P|} \sum_{x=\psi}^{N} C_x^N (1-\Pi(\mu))^x \Pi(\mu)^{N-x} \quad 4.20$$

To learn all of M correctly, the net should be constructed so that both Equations 4.16 and 4.20 approach 1.

For a given M, both $\Lambda(\nu)$ and $\Pi(\mu)$ increase as I increases. However, as I increases, the net size increases exponentially, and it is therefore not usually practical to set I very high. On the other hand, as ψ increases, $\Lambda(\nu)$ increases but $\Pi(\mu)$ decreases.

For systems with a large number of training patterns or nodes, it may not be trivial to solve Equations 4.16 and 4.20, even with the aid of a computer. However, an exact solution to these formulae is probably not necessary in the general case: the contribution of this section is more likely to be the understanding of how the parameter ψ affects likelihood of success of a given net on a given problem. In specific, if the system seems unable to learn to reject negative training patterns, ψ should be raised; if it does not consistently accept all of the positive training patterns, ψ should be lowered.

4.7. The Food-finding Creature

The proceeding sections of this chapter have described the design of a machine which is claimed to satisfy P1, P2 and P3: an exploratory automaton, which learns from a scalar reinforcement signal, and which can extrapolate from results which occur only at the end of a series of actions.

The classic testbed for exploratory learning automata is, by analogy with animals, a simulated creature which travels around its environment in search of positive reinforcers and avoiding negative reinforcers. This section considers such an ADB system.

The variant of the food-finding creature problem considered here differs from those mentioned earlier in this thesis in that the world is not a square or toroidal grid, but a state transition matrix. This means that knowledge of the current state can be used to predict future ones, in a manner which is impossible if the world con-

sists of a number of food objects scattered at random across the grid.

A set of M world states of locations exist, each providing a distinct 64-bit input pattern to the system. These patterns are given in Appendix A. Of these some $M^+ \subseteq M$ are positive while the remainder $M^- \subseteq M$ are negative. At each time step, the automaton is in some state $x \in M$, and has a choice of moving left, right or straight ahead; a transition matrix determines the next state: $T: x \in M \times move \rightarrow y \in M$. If $y \in M^+$, a positive reinforcement is supplied immediately (as if the automaton experienced the taste of food). If the automaton enters $y \in M^-$, there is an immediate positive reinforcement, followed by a negative reinforcement delayed by some $D = 4$ time steps (as if there was a taste of food later followed by nausea).

Thus the task is firstly to learn to predict the three adjoining states from the current one, and also to select the moves which result in some $y \in M^+$ and not $y \in M^-$, even though the results are delayed and contradictory signals may intervene. For example, negative reinforcement may not arrive until some time after $x \in M^-$ has been entered, and it may arrive just after some element of M^+ has been entered. After learning, all negative states should be transient: that is, the system should never execute a move which leads to a negative state. As many as possible of the positive states should be re-entrant: that is, the system should be disposed to enter positive states when a move from the current state leads there.

There are many complications. States which are themselves positive but which lead inevitably to negative states should be avoided. States in which one move leads to a negative state should be re-entrant as long as there is at least one possible move leading to a non-negative state. States entered after a negative one should be not be associated with negative reinforcement just because there may be negative reinforcement which arrives soon after they are entered.

The system used in this experiment was described in [Mye90] and is shown in Figure 4.9; it consists of two separate learning modules. The associator module AM

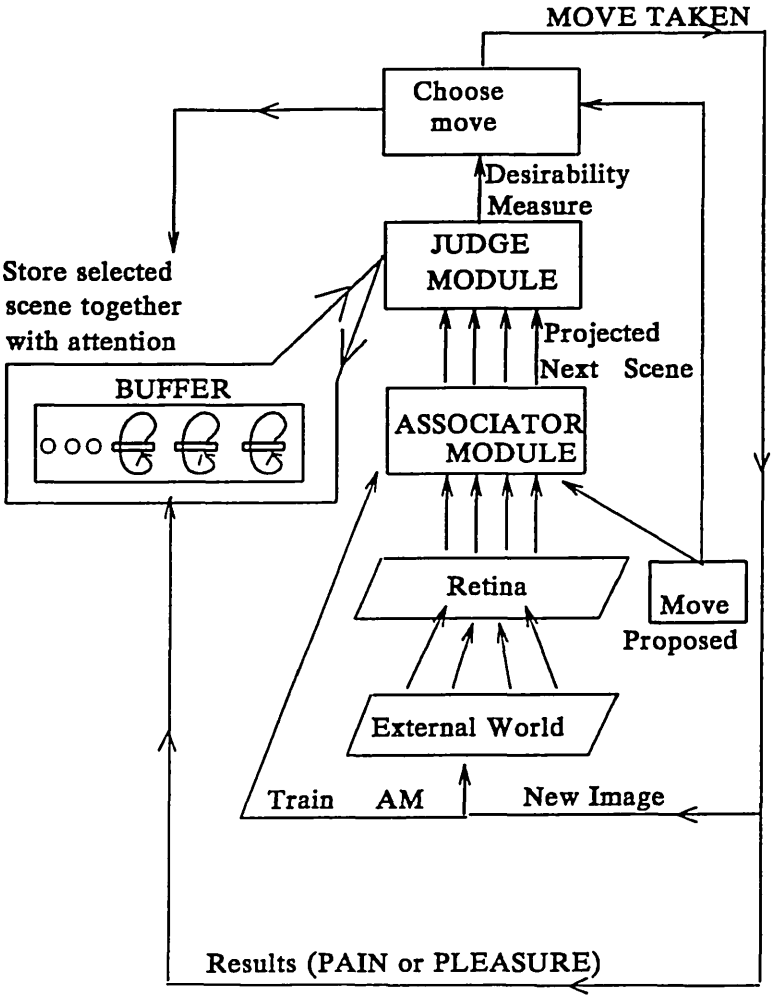


Figure 4.9. The system implementing a food-finding creature.

learns a mapping $AM: x \in M \times move \rightarrow y \in M$. Ideally, it should learn a function identical to the transition matrix T.

This AM is an MPLN network consisting of 64 pyramid structures, each outputting a single bit, and therefore the AM output can come to represent the pattern expected as input if a certain move is made. Each pyramid has as its bottom layer 8 MPLNs which each receive input from two bits describing a proposed move and eight bits mapped randomly from the current input pattern (each bit in the input pattern will be mapped to exactly one MPLN). These 8 MPLNs then feed into a single second layer node for each pyramid; all stored values are initialised to $s = 0.5$, and the output probability function is the steep sigmoid Φ_H^p of Equation 2.29.

After a move is selected and executed, the old input and selected move are clamped as input and the pattern representing the new state is clamped as output, and the addressed sv in each MPLN j in the AM is trained as:

$$\Delta sv = 10*r, \text{ if } y_j = 1$$
$$= -10*r, \text{ if } y_j = 0$$

where $r = -1, +1$.

Given the predictions of the AM, a second, judge module JM learns a mapping from state to desirability measure.

When the automaton enters a state x, the AM produces each of the three states it predicts will occur if the bug moves to the left, right or ahead (inactivity is not allowed). The JM processes each of these predictions in turn, and assigns a desirability to each. The output of the bug is the move which the modules predict will result in the highest desirability rating. Again, the JM is a network of MPLNs with stored values initialised to $\hat{s} = 0.5$ and outputting according to $\Phi_{\hat{s}}^{P}$, the soft output probability function of Equation 2.28. The desirability is simply the sum of the output nodes in the JM which output a "1" in response to the AM's prediction.

The predicted next state plus this move are stored in an attentionally-driven buffer, $n = 5$, where attention is set as $f(x) = n = 1.0$, and is decremented by $\delta = 0.2$. When results arrive, the JM is trained so that each state in the buffer is reapplied as input, and the addressed MPLN locations in the JM are updated according to the rule:

$$\Delta sv = \kappa_{r,} \text{ if } y_j = 1$$
$$= -\kappa_{r,} \text{ if } y_j = 0$$

where $\kappa_{+1} = 5$ and $\kappa_{-1} = -25$. This ensures that, if reinforcement is positive, desirability will rise; while it will fall if reinforcement is negative. Again, the negative increment is larger since there are fewer negative elements in M.

[Mye90] describes results with a JM consisting of 25 8-input MPLNs, mapping

randomly to the input pattern (and thus every bit in the input pattern was polled 3-4 times). The desirability was therefore a sum in the range 0..25.

The systems were trained by allowing them to traverse state space, with the appropriate reinforcements, until learning was such that the negative states would never again be entered. In this experiment, $M^- = \{A,B,C\}$.

After about 2,000 time steps, or about 24 negative reinforcements, the systems tended to find solutions, although not every positive state was re-entrant. Figure 4.10a shows one such solution: no negative states are re-entrant, and neither is state R which leads inevitably to a negative state; but positive states S, D, W, K, N, E, G are also not re-entrant.

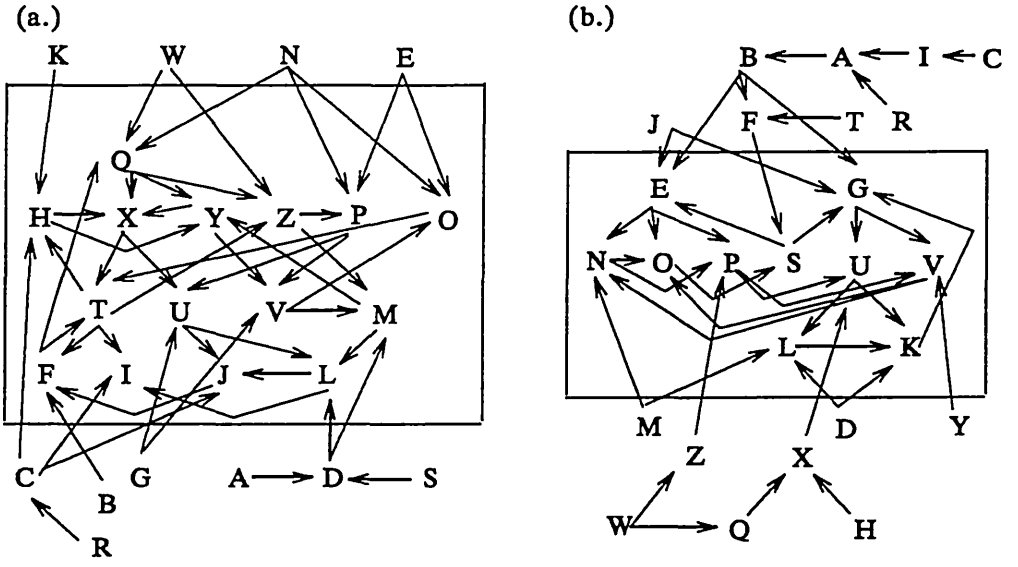


Figure 4.10. Example solutions found by the ADB food-finding creature: (a.) with AM and JM trained together, (b.) with pre-trained AM. Circled states A, B, C = negative; state R leads only to negative states; box encloses re-entrant states at solution.

In this solution, the system experienced 22 negative states within the first 1,000 cycles, and then only entered one negative state within the next 3,000 cycles, after which its memories were such that the negative states would never again be entered.

By training the AC first and separately, the system learns faster. The JM is then getting perfect information about the next states; when the two are trained

together, the JM is trying to judge desirability from predictions which are themselves only being learned. The AM can be trained to mimic the state transition T perfectly within some 2,000 cycles; the JM then learns its mapping perfectly within only 500 cycles. Figure 4.10b shows such a solution. Although this solution was found more quickly, it is actually not as good as that found in Figure 4.10a, since more positive states are not re-entrant. Quick learning means more pathways are eliminated quickly, among them transitions into positive states.

In this example problem, there are relatively few patterns to be learned, each uniquely specifies the state, and the transition matrix is deterministic. The chief difficulty lies not in the problem itself but in the learning conditions: the learning must be on-line, the patterns do not all necessarily appear with the same frequency, and the goal, ideally, is only very vaguely specified.

A second important aspect of this problem is the necessity for second-order prediction. Some states in the environment ("cul-de-sacs") are themselves elements of M^+ but lead only to states which are elements of M^-. That is, although a state may be positive, it may entail negative reinforcement. The ADB system is able to learn this backward association and come to avoid the cul-de-sac state. This is a consequence of the presence in the buffer of previous states when a reinforcement arrives.

Finally, in this formalism, the system has been restricted to a single "drive". There has been no distinction between different elements of M^- and M^+, and no internal goals which cause the system to alter its behaviour at various times. It would be possible to consider multiple goals by creating several "drive" monitors, each of which feed into the JM and thereby affect the subjective desirability of each state at a given time. Jordan [Jor86] and Sharkey [Sha89b] have considered architectures like this, where the input is a combination of external information and internal state or context.

4.8. Conclusions and Summary of Chapter 4

In this chapter, the ADB model was introduced. It consists of an associative memory whose inputs and corresponding outputs are buffered until reinforcements arrive. Because the buffer is of finite size, each new input/output pair is assigned an attention; if this attention is stronger than that of some element in the buffer, the new pair ousts it and takes its place. Attention is usually related to the unpredictability of an input/output pair: if the results are well known, there is less need for the pair to enter the buffer.

The several parameters which describe an ADB system were subjected to analysis. The most important results are these:

- The maximum delay over which learning is possible depends on attention-assigning function and memory increment, but not on buffer size.
- The maximum delay over which response to a pattern converges to 100% (or 0%) is also independent of buffer size.
- In an exploratory context, the MPLN stored values should be initialised so that there is a high probability of outputting 1 to a random pattern; useful positive and negative memory increments can be derived as functions of the maximum possible attention and of the proportion of inputs which are positively reinforced.
- The number of MPLN 1 outputs needed for a system accept should be raised to eliminate false accepts of negative patterns; but if raised too high, the system will begin to fail to accept positive patterns as well (this is a feature of all discriminator-structured networks).
- The buffer may be maintained at a system level, or locally at the nodes, allowing for parallel updates.

An example food-finding creature was described that could learn when results were delayed, interleaved, and even when actions were themselves positively reinforced but should be avoided as they lead inevitably to negative reinforcements.

It is important to note that as the problems scale up, in number of nodes, number of training patterns, complexity of training patterns, etc., the associative memory may have to become more sophisticated, but the ADB paradigm remains unchanged, as it is defined, in the first case, independently of any such variables as N, $|M|$, etc. Section 4.6, which discusses some of these issues, holds for any such MPLN network, whether or not it is embedded in an ADB system.

If the reinforcement delay were not some constant D, but rather some time in the range $0..D$, the behaviour of the system would not change overmuch. The principal effect would appear in Equations 4.2, 4.3 and 4.4, where the attention-setting function f and memory increment δ would define the maximum average expected delay $\overline{D}$. The simulations described in [Mye89b] and [Mye89c] learned successfully when positive reinforcement was immediate, and when negative reinforcement occurred with a delay randomly scheduled 0..4 time steps later.

If the reinforcement delay is reduced to $D=1$, then the system becomes more like those prediction-as-reinforcer systems described in Section 3.5. It still has the advantage of being able to perform backward learning, as various previous states will still be present in the buffer when this reinforcement arrives; if there is no correlation between earlier states and the current one, this may slow learning somewhat as the earlier states receive spurious reinforcement.

There are several critical aspects of ADB systems which are not discussed in this thesis, and which form major portions of future planned research. Most notably, in all of these simulations, the attention-setting function is the critical parameter, and yet it is selected in an essentially arbitrary fashion – either to be a sine curve, or else as a plateaued step function, if this simplification does not seem costly

in terms of system performance. The overall shape of the function is clearly required to be low or zero for high and low vote tallies and maximal for intermediate tallies, but there is a wide range of curves which satisfy these criteria. Perhaps the selection mechanism which is most in tune with the philosophy of neural network research is for this curve to be fitted experimentally by the system itself. This would almost certainly have a considerable cost in terms of learning time, since the basis on which the system assigns buffer space would be changing even as the responses to the buffered items change based on their attentions.

Another very important extension will be considering non-trivial decay schedules for buffered items' attentions. In human short-term memory, items are forgotten with time; but they can be maintained for longer if there is a conscious effort made to "remember" them (I "remember" where I parked the car today, but will have forgotten this datum by tomorrow in time to store the new parking space) or if rehearsal takes place (I repeat a friend's phone number to myself to maintain it until I am finished dialling the telephone, and then quickly forget it). Both of these phenomena are analogous to items being maintained in the ADB buffer for long periods, without any necessary effect on the long-term (MPLN) memories associated with them. The ADB system would be much more elegant if provided with capabilities for rehearsal, longer retention of items which are particularly relevant or which a higher controller instructs should be maintained, and even the ability to do "one-shot" learning – memorisation after a single exposure to critical data. However, these effects seem to depend on the existence of some higher level controller to dictate when they should occur, and such a higher level controller is beyond the scope of the topics considered here.

Finally, it is important to sum up the essential differences between the ADB system and related work. It differs from simple buffer-maintaining systems in that it can learn over delays longer than the buffer size. It differs from eligibility-

maintaining systems in that it can reinforce temporally distant events strongly when necessary, but is not required to do this in every instance. It differs from temporal difference and AHC methods in that it is not tied to associate the effects of an input at time t with those at $t+1$ and $t-1$, and also in that it can learn about sequences which involve different reinforcement to a state depending on which states preceded it.

In the chapter which follows, an ADB system is designed to simulate portions of the octopus visual attack learning centre. The heuristics and rules derived in this chapter serve as troubleshooting aids: their primary use seems to be as guides for how to improve the behaviour of a system.

CHAPTER 5: A MODEL OF THE VISUAL ATTACK LEARNING SYSTEM OF OCTOPUS VULGARIS

5.1. Introduction

The learning automata considered in this thesis have three major properties. They are exploratory rather than passive (and hence they have no discrete training period) (P1). Reinforcement does not come in the form of a desired response but in the form of a global estimation of success (P2). This global success measure may not occur immediately; it may only become available at the end of a sequence of arbitrary length, and other reinforcements may arrive in the meantime (P3).

These criteria are not arbitrary, but are conditions which even very simple animals must overcome in order to survive. Any man-made learning system claiming some degree of intelligence must reasonably be expected to overcome these conditions as well.

P1 and P2 obviously apply to animal life. P3 may not at first glance appear relevant, if one considers animals which do not seem to be capable of learning extended problem-solving sequences. However, delay learning is actually characteristic of all operant conditioning tasks:

> "After an object has been detected by distance receptors, such as those of vision or touch ... there is an interval during which the animal moves towards the object in question (or draws it in) before the 'reward' [e.g., taste] arrives. During this interval the classifying system must somehow retain information about the characteristics of the object that evoked the approach." [You71, p.246]

Not only must the original information bridge this time gap, but it must also persist in spite of intervening patterns. In the above example, the intervening visual stimuli will include an arm reaching out and obscuring the object, and a hugely enlarged version of the object as it comes close to the body. Eventually the stimulus will no longer be visible at all as it is drawn into the mouth. Only then do (pleasant

or unpleasant) taste sensations arise.

This chapter examines the performance of a neural network system using attentionally-driven buffering (ADB) in tasks designed to be like those of which a simple animal is capable.

Eric Kandel, in his *Cellular Basis of Behaviour* credits genetics student Chip Quinn with the following specification of the neural scientist's ideal organism:

> "The organism should have no more than three genes, a generation time of 12 hours, be able to play the cello or at least recite classical Greek, and learn these tasks with a nervous system containing only ten large, differently colored, and therefore easily recognizable neurons." [Kan76, p. 45]

Though whimsical, this imaginary animal illustrates the paradox (for cognitive modellers as well as neurobiologists) in dealing with real animals: either the nervous system is so complex as not to be very well understood, or else it is so simple that the creature is not capable of any sophisticated adaptive behaviour.

The animal chosen for the purposes of this chapter is in some sense intermediate. *Octopus vulgaris* Lamarck has a much smaller brain than man, but it has been extensively studied and shown capable of a considerable number of discrimination tasks. Although the cephalopod brain differs in many ways from the mammalian brain, it is likely that at least some of the same operating principles may hold.

The remainder of this chapter is organised as follows. Section 5.2 briefly overviews the salient features of the visual attack learning system in *Octopus*. Section 5.3 then describes the ADB-based system which is to simulate the behaviours of the octopus:[1] OVSIM (for Octopus Vulgaris SIMulation). The two sections that follow compare and contrast octopus and OVSIM in discrimination learning tasks, normally and with ablation of some sections of the brain. Finally, Section 5.6 summarises the chapter, compares OVSIM with a biologist's model of the octopus brain, and

[1] Throughout the remainder of this thesis, the term octopus will be used to refer to members of the particular species *Octopus vulgaris*, except as otherwise noted.

examines some predictions about *Octopus* which the ADB approach suggests.

5.2. The Visual Attack Learning System of *Octopus vulgaris*

Octopus vulgaris Lamarck, the common octopus, is a species of cephalopod common near the beaches of Naples, and whose nervous system is well understood due to extensive neurobiological and behavioural studies. Its main food source is crab, and laboratory specimens can be trained to attack arbitrary simple figures when rewarded with bits of crab or to eschew others when punished with mild electric shock. Trainable stimuli include black and white plastic geometric shapes [You64, p. 79 and 128-37], sinusoidal gratings [MuG88], and even crabs [You64, p. 82]. Thus the animal can be taught to respond in ways which demonstrate conditioned learning.

Anatomically, the octopus has a reasonably sophisticated nervous system consisting of some 500 million nerve cells. The particular suitability of the octopus for this sort of study derives from the fact that the motor processes and sensory processes are largely kept separate, and over 300 million of these neurons are distributed among the arms, where they control delicate movements and intricate somatosensory explorations. The ganglia overseeing reflex reactions are located in the arms as well. Thus, only some $2x10^8$ nerve cells are centralised; even within this central area, some 50 subregions form well-defined lobes, with limited interlobe connections.

A further advantage of studying the octopus is that its behaviours are quite stylised. *Octopus vulgaris* lives in crannies on the ocean floor, or homes made from piles of bricks in an experimental tank. From the home, an arm will be put out to seize a crab swimming by. If the crab is out of immediate reach, the octopus may jet out, seize the prey and return to cover. The captured meal is paralysed with a salivary secretion, broken up by beak and radula, and passed into the mouth. Life

for the octopus consists in large part of repeated decisions to attack moving objects in the visual field or to withdraw from them.

Other behaviours occur in these animals, including mating, posturing at rivals, conditioned responses to touch stimuli, and complex skin patterning for camouflage or to accompany courtship displays and defence attitudes. Some species of octopus even engage in complex social organisation [MaL85] and migrations in search of homes [HAR84]. Only conditioned learning about attack decisions for objects in the visual field will be considered here.

5.2.1. Trainable discriminations

As described above, the octopus does not lead a particularly varied life, and the obvious and usual experimental set-up is to condition the octopus to accept (grasp and draw in) one stimulus and reject (release, refuse to grasp, or even flee from) a second. The usual reinforcement for accepting the positive stimulus is a bit of crab or fish, and a mild but painful electric shock usually follows acceptance of the negative stimulus.

Little is known about newborn octopuses, as most laboratory animals are captured and thus have already been exposed to experiences for some time. Probably, the octopus has some dendritic arrangements, and hence behaviours, set up by heredity [You64, p. 138]. For example, the naive (untrained) octopus has an innate tendency to attack moving objects in the visual field, but it will seldom attack stationary ones [MuG88]. Usually, the initial response to a strange moving object is attack after a considerable delay; if the object turns out to be edible, the delay quickly disappears, while without positive reinforcement, hesitation becomes even more prolonged and attacks eventually die away altogether [You65a]. Within ten trials, according to Wells [Wel68], a normal octopus can learn to accept one neutral stimulus rewarded with food and to reject another neutral stimulus followed by

shock. Neutral objects tested include black and white geometric shapes [You64, p. 79, 128-37], objects at different contrast [You68], and sinusoidal gratings of at least discriminable frequency [MuG88]. Maldonado [Mal63] describes experiments where octopuses were conditioned successfully to discriminate on the basis of movement, brightness, direction of movement in relation to the long axis of a geometric figure or in relation to the points of the shape, territory, vertical or horizontal extent of a figure, and analysis of contour. Wells notes [Wel68, p. 167] that octopuses "have elaborate eyes and can learn to distinguish most of the differences between shapes that are apparent to us" – although they are apparently not sensitive to colour [You64, p.113].

A non-neutral stimulus is one where the animal shows some innate predisposition to attack or reject – for example, Young trained octopuses to reject crabs, their natural food source [You64]. The animals can also be taught discriminations involving non-neutral stimuli, such as to attack a crab but reject one paired with a white plastic square [BoY55].

However, even after long training on any of these tasks, performance is never perfect (c.f. [MuG88]); the trained animals show occasional failures to accept positive stimuli, and mistaken attacks on negatively reinforced stimuli.

Finally, if one arm of a blinded octopus is taught a discrimination, the learning eventually (over a course of hours [Wel59a]) spreads to the other arms. The same transfer is true of visual learning concerning objects presented to a single visual field and later tested in the other.

An important inability of the visual learning system is a lack of generalisation over size. At least in the octopus, size generalisation is not conferred automatically during learning, so that an object at one size or one distance may not be recognised if enlarged, or moved significantly further away. The fact that an octopus knows to attack a crab wherever it appears in the visual field appears to be a result of learning

with the crab figure at various retinal sizes during the octopus's approach to the food object. Young [You64, p. 171-2] has shown that a trained animal does not recognise a stimulus when doubled or halved in size, and is even less likely to recognise the object when the viewing distance is altered significantly.

5.2.2. The functional organisation of the visual learning system

The optic lobes, two structures lying just behind the eyes in octopus, receive as input the signals from the contralateral retina via the optic nerve (Figure 5.1). They send outputs to the motor system – initiating attack and retreat actions and also participating in a visual feedback loop with other centres to fine-tune motor actions. In addition to their role in generating motor responses, the optic lobes are thought to be the chief region of visual memory storage in *Octopus*. Together, the optic lobes contain some 92% of the nerve cells in the central nervous system [You64, p. 108].

The outer layer of the optic lobe receives the retinal information in a relatively unprocessed state, and is thought to consist of cells which classify visual input in terms of features such as lightness, movement, etc. [You71, p. 447]. The dendritic fields of these cells are regular in shape and mostly oval [You71, p. 68], and the cells may therefore act as shape and orientation-sensitive filters; most are horizontally or vertically elongated, and in fact, the animals do seem to find discriminations based on horizontal and vertical orientation to be particularly easy to learn [You71, p.470].

The cells of the optic lobe interior receive input from these classifying cells, but show no distinct fields, field shapes or topographic mapping [You71, p. 476]. They therefore receive information about widely varying areas of the visual field. These are the cells which, it is supposed, adapt their output appropriately to visual patterns. They often show two or more axons, which then leave the optic lobe for the motor centres to initiate attack or retreat actions.

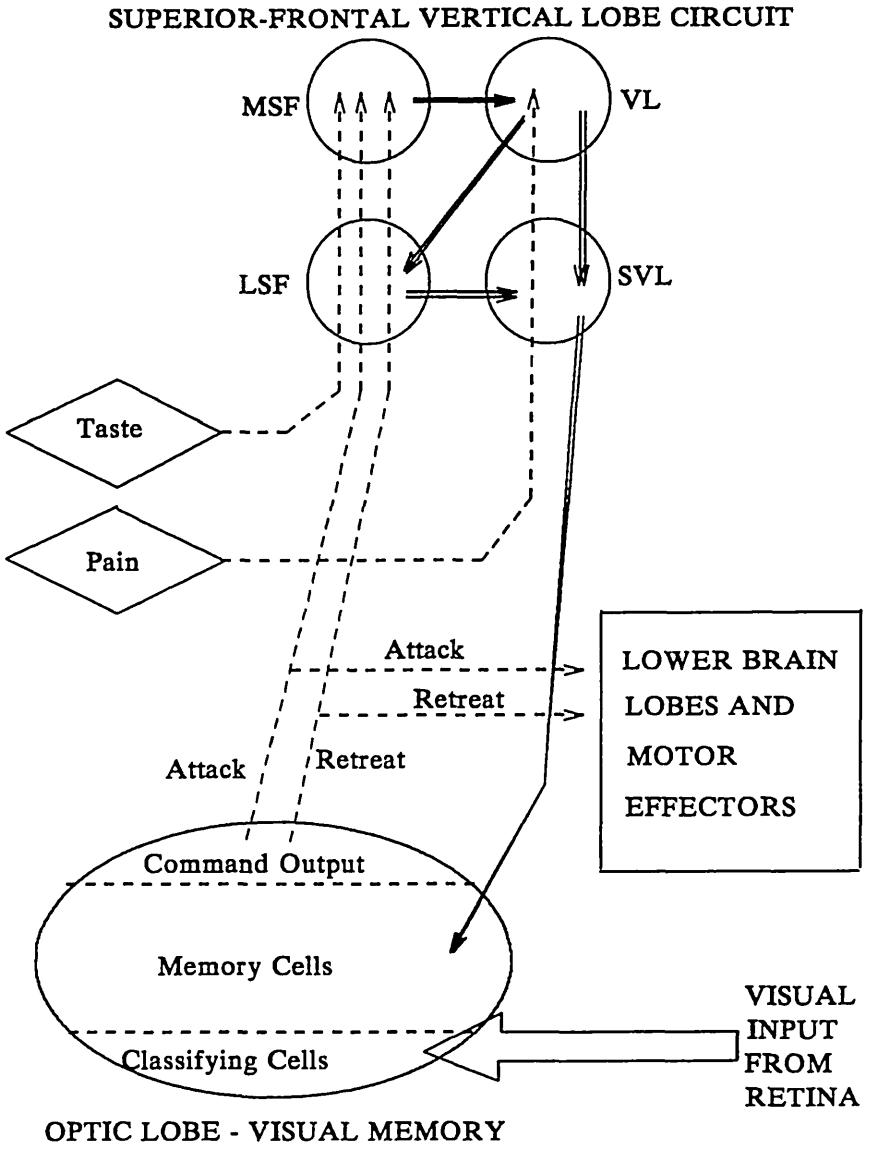


Figure 5.1. Schematic drawing of the visual attack learning system in Octopus, including optic lobes and SFVL circuit (adapted from [Boy67], [Wel59b] and [You71, p. 245 and 196]). MSF=Median superior frontal lobe; LSF=Lateral superior frontal lobe; VL=Vertical lobe; SVL=Subvertical lobe.

Vision in the octopus is basically monocular, and each optic lobe gives rise to an independent motor command to attack or retreat. The decision from the two lobes may conflict, and this conflict must be resolved either in the motor centres or elsewhere.

In addition to the motor centres, the output of the optic lobes also travels to the higher lobes: the median superior frontal, lateral superior frontal, vertical and

subvertical lobes. Together, these four lobes form the superior frontal-vertical lobe (SFVL) circuit which contains cyclic pathways among the four lobes and also with the optic lobes. Additional input to the SFVL circuit comes from sensors for taste and pain: the "results" of an executed decision to attack.

The circuit combines taste and pain (positive and negative reinforcement) to suppress or amplify tendencies for the optic lobe to signal attack [You71, p. 320]. It also aids in the setting up of optic lobe memories according to the reinforcement of taste and pain [You64, p. 203]. Its own output does *not* seem to be altered by experience.

Most information about the specific functioning of the lobes comes from ablation experiments: an animal, naive or trained to perform some discrimination, has one or more lobes removed, and the function of the missing areas can be determined by the resulting behavioural changes. The effects are often complex, as other factors come into play such as postsurgical shock and loss of input to otherwise intact nearby brain regions. In particular, any ablation of SFVL tissue interrupts both a self-reexciting chain and also a loop with the optic lobe: thus there may well be loss of function only indirectly related to the ablated tissue.

Consensus is that the superior frontals serve to increase the tendency to attack, particularly for distant objects which the octopus has to jet out to reach (c.f., [You64,p.202] and [You70]).

The vertical lobe is thought to serve an opposite function: to integrate the effects of pain, and where appropriate, use them to repress attacking. Its removal leads to some complicated effects. The primary effect seems to be the inability to learn avoidance behaviours. Learned responses are maintained intact – the octopus will still avoid objects it previously learned not to attack – but it cannot learn not to attack a new object associated with electric shock [You64, p. 212 and 214], particularly if it is an object the octopus is innately predisposed to attack, like a crab

[You70]. Removing the vertical lobe also results in a reduced level of nonspecific tendency to attack [You70].[2]

The subvertical lobe provides the system's feedback output to the optic lobe. It combines the amplification and suppression of the other lobes: in effect, it passes through the superior frontal signals unless the vertical lobe intervenes with pain signals [You64, p. 203]. Again, a principal effect of ablating this lobe is a reduced tendency to attack [You71, p. 320].

There is no evidence [You64, p. 230] that the centres of the SFVL circuit learn with experience or alter with satiation or hunger. Instead they seem to deal with changes in the tendency to attack any object, and to help assign reinforcement information to memories within the optic lobes.[3]

It seems likely that the long-term visual memories are stored within the optic lobe. The SFVL circuit may maintain short-term memories until results occur and then pass these memories on to the optic lobe for permanent storage there. Alternatively, short-term memories may be stored within the optic lobe, but the SFVL may be responsible for setting up these short term memories or for overseeing their eventual transference into long term memory.

5.3 Details of OVSIM - The Octopus Simulation

A system to simulate learning of discrimination tasks like those performed by the octopus has been built around the ADB concept developed in Chapter 4. The

[2] It is suggestive that the vertical lobe, which may be the SFVL lobe most implicated in learning, is also the only one to contain many amacrine cells – in fact these integrating neurons make up most of its bulk [You64, p. 207]. Amacrines have been implicated elsewhere in learning; Wells and Young [WeY65] have shown that at least a few thousand of these cells are necessary for (touch) learning to take place. However, their exact function remains a mystery (J. Z. Young, personal communication, May 1990) – in *Octopus* and in the brains of other animals.

[3] It is believed that the SFVL circuit is also involved with the transfer of learned knowledge from one to the opposite optic lobe. An octopus with training in one visual field, and hence in one optic lobe, can solve the same problems on the other side [You64, p. 168]; an animal without the vertical lobe cannot do this. In touch learning, experience with a single arm in a blinded octopus will be likewise distributed to the other arms [Wel68, p. 172], but the transfer may take up to an hour.

Octopus vulgaris simulation, OVSIM, buffers "visual" images which may be "attacked" until the arrival of "taste" or "pain" results.

The OVSIM system is illustrated in Figure 5.2, and may be compared with the generalised ADB systems shown in Figure 4.5: the non-adaptive classifying cells and adaptive memory units of OVSIM represent the associative memory of the ADB system; OVSIM's attention-setting and output-generating units match those in the ADB system, and it will be claimed later that they may correspond to the SFVL circuit in *Octopus*.

5.3.1. Stimuli

OVSIM was designed to learn discrimination tasks resembling those to which Young, Wells, Boycott and others trained *Octopus*, and in which the reward (food or pain) only occurred after a sequence of attacking. In OVSIM, as in *Octopus* work (e.g., [You58a]), a *trial* is defined as the presentation of a stimulus and terminates when reward is given (if positive), punishment is given (if negative), or a certain time limit expires in the absence of attacks. For the animal, reward is a small bit of fish (or if the stimulus is a crab, the animal may be allowed to eat this directly) and punishment is a mild shock. In OVSIM, reward and punishment are activation of dedicated channels, which affect the stored memories as described in Section 5.3.5.

OVSIM stimuli consisted of binarised patterns, each 32x16 bits in size, allowing for simultaneous presentation of two 16x16 stimuli. The complete set of stimuli used are shown in Appendix B. The patterns were constructed to resemble the stimuli used with the octopus, as this simplifies later discussion and because these sorts of patterns contain the features which the first level of OVSIM visual processing was designed to pick out (see Section 5.3.2). Of course they contain no semantic value for OVSIM – other than that learned during training.

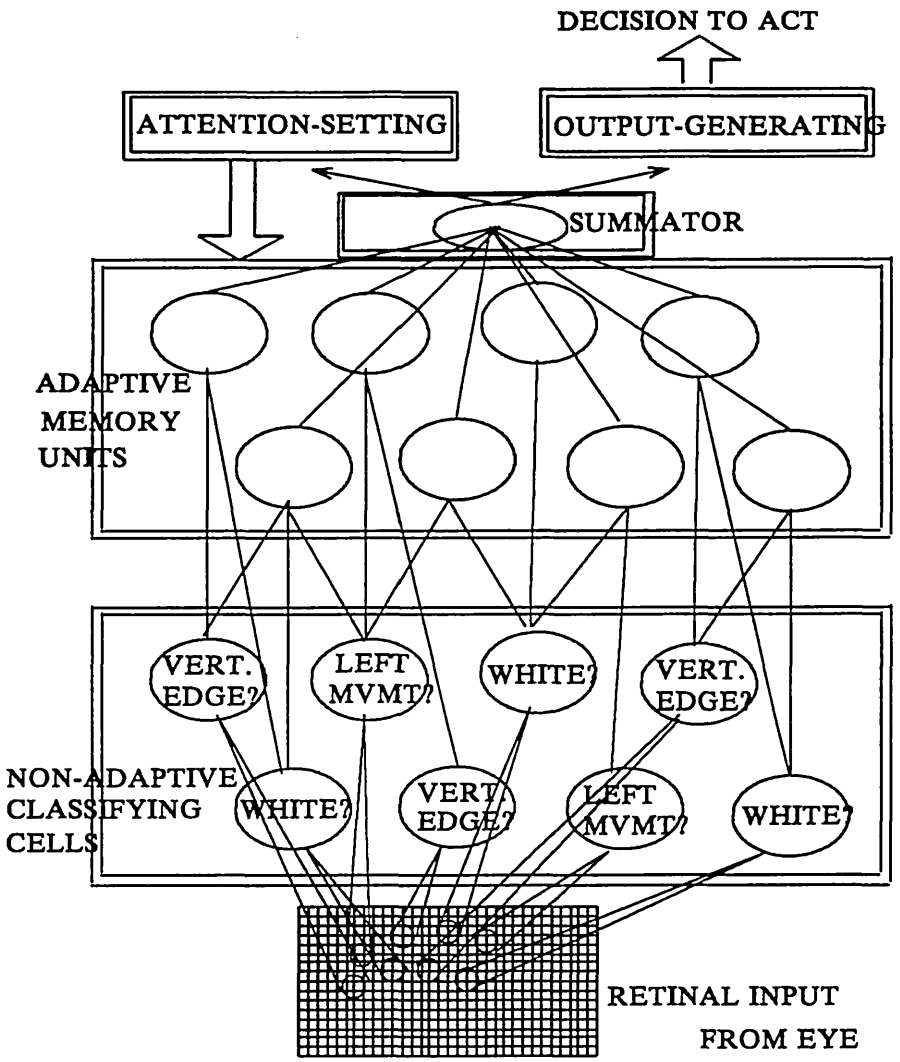


Figure 5.2. The OVSIM system. 4200 classifying cells perform feature-extracting functions; this recoding is passed as input to the 1615 adaptive memory units. The summed output of these adaptive memory units is transformed into a binary act decision as well as into an attention measure, governing the placement of the current input pattern into the ADB buffer.

In *Octopus* experiments, geometric stimuli were often plastic figures measuring about 10 cm^2 (e.g., [You58a]), approximately the same size as the crabs. This convention was followed in OVSIM stimuli, as Appendix B shows.

One OVSIM trial consists of one or more *cycles*, each consisting of a presentation of the input pattern to the associative memory, a system output, and the response of the world to this output. A trial may last up to 20 cycles if no attack occurs; after this the trial is terminated and scored as a no-attack. If an attack

occurs, the (positive or negative) reinforcement arrives 4 cycles after – during the interval, the visual input changes to a series of patterns representing the extension of an arm to the object pictured, the drawing of this object downward as if under the mantle, and finally the disappearance of arm and object. The arrival of reinforcement terminates the trial. Example sequences following attack on a positive and negative pattern are shown in Appendix C.

In octopus training, stimuli are jiggled by hand at a rate of 3x per second to ensure movement-detecting cells in the octopus eye or optic lobe are stimulated.

In OVSIM, patterns are jiggled one bit up or down on each cycle to simulate the effects of movement. Positive and negative trials are alternated (there is no evidence that *Octopus* can learn to discriminate on the basis of sequences [WeY65], and indeed this version of OVSIM cannot do so either).

Octopus experiments have involved trial lengths as short as 15 seconds [BoY56] or 30 seconds [You58a]; a more usual trial length is 2 minutes (e.g. [BoY55]). If 2 minutes in *Octopus* is to be equivalent to 20 cycles in OVSIM, then 1 OVSIM cycle represents about 6 seconds of stimulation. Young [You58a] and others leave at least 40 minutes between trials. The Unspecific Effect which follows reinforcement is known however to persist 1-2 hours (see Section 5.5). Accordingly, except as otherwise described, OVSIM trials are separated by 1200 cycles, or an effective 2 hours. There is no input during these periods.

As mentioned above, in laboratory conditions the negative stimuli are usually left visible even after an attack, allowing the animal to make a further attack if time (and inclination) allow. In OVSIM an attack ends the trial. This decision was made for two reasons. First, allowing multiple attacks per trial means that number of trials does not accurately reflect number of learning experiences. Second, leaving the negative stimulus visible after reinforcement leads to confounding effects in conditions of damaged learning systems, as will be discussed in section 5.5.

5.3.2. Classifying cells in the optic lobe

The first layer of processing in a visual system generally recodes visual information into feature space. It is known [Mal63] that *Octopus* can discriminate shape, movement, brightness, contour, vertical or horizontal extent. There are no neural cells besides the receptors in *Octopus* retina [HaL84] – whereas vertebrates have 5-6 levels of processing retinal cells – and the processes of these receptors form the optic nerve. It is likely that feature detection takes place in the optic lobe.

Even though the exact types of classifying cells in *Octopus* are unknown, there are certain features which trigger the visual systems in most species. Blakemore lists these as: contrast or edges, movement, direction of movement, convexity or size, orientation of edges, and overall illumination [Bla75]. Therefore, the feature detectors in OVSIM were designed to extract these features from the input patterns.

OVSIM's front line of processing consists of Classifying Cells (CCs), which search for the existence of predefined features.[4] These cells cover overlapping 3x3 bit areas of the retina, and output "1" if the feature is present and otherwise output "0". There are 10 types of CC implemented, detecting whiteness ("0"s), blackness ("1"s), on-surround patterns, off-surround patterns, vertical edges, horizontal edges, and movement upward, downward, left and right. The patterns to which each type of CC respond are shown in Appendix D. 420 9-bit fields exist on the 32x16-bit input patterns, each covered by one of each type of CC; hence 4200 CCs exist in total.[5]

The CCs are non-adaptive, and are implemented as RAMs which store a "1" at locations addressed by patterns representing the desired features, and "0" elsewhere. Mammalian cells in early somatic and visual cortex with feature-detecting function

[4] The general topology and nomenclature of the OVSIM Optic Lobe follows from Maldonado's model of the octopus learning system [Mal63] which is compared with OVSIM in Section 5.6.2.

[5] In the simulation, only a single one of each type of CC exists, and is "moved" across the input pattern, and its response at each location is noted.

(like those the CCs emulate) self-organise during a critical or sensitive period in infancy, after which their functions remain relatively fixed [Bla88 p. 37-39].[6] This seems logical; higher levels must learn discriminations based on the feature encodings of early levels. If the output of early levels is subject to frequent change, the information stored in higher levels will be quickly out of date unless it is constantly updated to conform. The simplest solution seems to be to hold the transforms executed by the early layers to be constant.

5.3.3. Memory cells in the optic lobe

Maldonado [Mal63] proposes that the octopus's optic lobe outer layer first performs feature extraction (as simulated by the CCs), and that its inner layer of cells learn and store appropriate responses to patterns of CC activity representing aspects of the stimulus.

These adaptive cells are represented in OVSIM as Memory Cells (MCs). MCs are units with 6, 8, or 10 inputs from the CC outputs; enough of each sort of MC exist to cover all of the CC outputs with 98% probability: there are 686 6-input MCs, 515 8-input, and 414 10-input, for a total of 1615. Connections from CC outputs to MC inputs are random but fixed throughout a simulation.

Each MC is implemented as an MPLN augmented with a buffer. The binary MC input X forms an address into the MPLN memory, accesses the stored value at that location sv_X, and is transformed into binary output according to the output probability function:

$$\Phi^P(sv_X) = sv_X \qquad 5.1$$

The summed output of all 1615 MCs is used to generate an output action for OVSIM

[6] The somatic cortex of adult animals can certainly reorganise – for example an adult monkey trained to a fine pointing task will develop an enlarged cortical representation for the active finger [Bla88 p. 138], but this is on a much smaller scale than the changes during the critical period.

and to generate an attention value.

The buffer associated with each MC has capacity to store two of the previous addresses to that MC. When reinforcement $r=-1,0,+1$ arrives, each of the buffered addresses with non-zero attention is reapplied to the MC's MPLN, and the value accessed sv_X is adjusted as:

$$\Delta sv_X = rH \qquad 5.2$$

Section 5.3.5 defines $H=1$.

It remains to define $\hat{s}$, the value to which the sv_X should be initialised before training. If $\hat{s}=0.5$, approximately 50% of the 1615 MCs will output "1" in response to the first pattern – for a total of about 808. If the output-generating function is as described in Section 5.3.4, then the probability of attack on a given cycle will be 0.15, while the probability of attack on a given trial (20 cycles) will be 0.96. This ensures that most new figures will be attacked on their first presentation.

This agrees with the observation by Boycott and Young [BoY56] that "an octopus trained to attack crabs usually comes out to attack the figures used here on their first presentation."

5.3.4. Output generation and assignment of attention

The output from the MCs may be summed to yield a number, V, in the range from 0 to 1615, indicating a strength or confidence in the decision to attack. To this sum is added UE+ and UE-, the current unspecific tendencies to attack or retreat (see Section 5.3.6), and the total is then transformed into a probability of attacking as shown in Figure 5.3a, and thence into an all-or-none decision to attack.

In *Octopus*, there are two competing decisions from attack learning and retreat learning subsystems; ideally, when one votes to attack, the other votes to "not retreat", and so forth. When conflicts do arise between the outputs of these two sys-

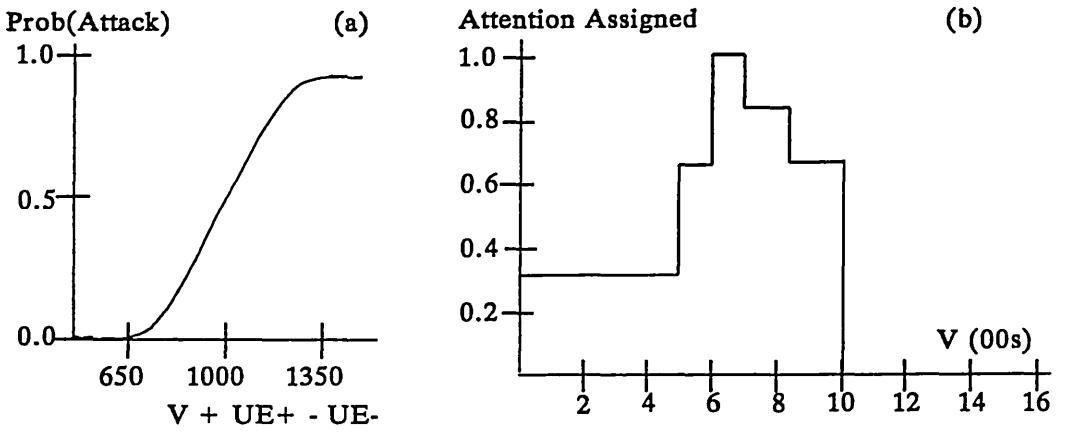


Figure 5.3. The functions used by OVSIM to (a) generate output from summed votes and (b) assign attention from sum of votes.

tems, some mechanism such as lateral inhibition must resolve the dilemma. The retreat learning subsystem does not form part of this study, and so the competition and inhibition are not considered further here.

At the same time as V, the total number of votes, is transformed into a decision whether to attack, it is also used to assign an attention to the current input image, as shown in Figure 5.3b. Attention is highest for $600 \leq V < 850$ – the most random responses from the system. It decreases as V becomes more polarised – indicating confidence that attack will be rewarded or that it should be avoided. Once the results of an action are sure, there is little more to learn about that stimulus, and it is not so important that the system retain information about it against arrival of reinforcement. The curve is skewed so that inputs which generate a low probability of attack always also generate non-zero attention – as, if an attack occurs, there is obviously still more to be learned if they are to be avoided, while inputs with a high probability of attack generate zero attention so as not to take up buffer space.

Each MC contains a buffer with capacity to store two of its previous input patterns together with the attention for each. If the new, current input is assigned a higher attention than one of the buffered elements, then it overwrites that element. Attention decays linearly on each cycle, and so even if never overwritten, elements

may only stay in the buffer for a finite period while their attention is non-zero: the decrement to attention is $\delta = 1/6$, and so the maximum delay bridgeable is 6 since for all MCs i, $f(R_i(0)) = 1$.

5.3.5. Attack and reinforcement

When OVSIM decides to attack, the action is treated as indivisible and irreversible. This is a simplification, of course, but as Section 5.6.3 discusses, there is some evidence that attack is also irreversible in *Octopus*.

When an attack is made, the input sequence reflects this activity. In the case of the octopus, the animal moves toward the stimulus, grasps it and (if positive) draws it in and eats it. All of this provides input to the visual learning system which is essentially irrelevant: the important image with which to associate reinforcement is the original stimulus presentation. The OVSIM input, on attack, initiates a stylised sequence in which an "arm" pattern is superimposed on the stimulus pattern, and both are drawn down and out of sight, as if they had moved under the mantle towards the mouth. At this point, positive or negative reinforcement is administered. If negative, the object is then released and returns to its previous position; if positive the object remains out of sight as if ingested (see Appendix C).

In the octopus, negative reinforcement occurs immediately upon touching the electrified stimulus, while positive reinforcement might derive from the taste sensations, or even the visceral signals indicating food in the gut. In OVSIM, both types of reinforcement occur at the same delay: which is in some sense a compromise between the two extremes. The negative case is made harder by not allowing immediate reinforcement. The positive case is made easier in that only a small number of discrete images occur before reinforcement.

Reinforcement entails setting a quantity, H, to $+1$ or -1 for the next 10

cycles, or about 1 minute of simulated time. After this period, H reverts to its default value of 0. Its effects can be seen in the MC learning rule of Equation 5.2; so each pattern in the buffer can receive a maximum of $+10$ or -10 reinforcement. In order that all are more likely to remain in the buffer for the full reinforcement period, attentions of all buffered input images are boosted by a factor of 10 when reinforcement is detected.

5.3.6. The Unspecific Effect parameter

Section 5.5.3 will discuss the Unspecific Effect: an animal is more likely to attack an arbitrary figure after feeding and less likely after receiving a shock. This is in addition to the learning about the specific stimulus-response pair which occasioned the food or shock. This unspecific tendency decays logarithmically and disappears within 1 or 2 hours.

In OVSIM, there are two competing forces – UE+ which is an unspecific tendency to attack after receiving positive reinforcement, and UE-, which is an unspecific reluctance to attack after receiving negative reinforcement. They are scalars: UE+ is added to and UE- subtracted from V, the total votes. The result is then converted into a probability of attacking as shown in Figure 5.4b.

At rest, $UE+ = 300$ and $UE- = 200$, so that there is a slight tendency to attack even in the absence of any reinforcement. This is supported by the fact that upon removal of the brain centres in *Octopus* believed responsible for the Unspecific Effect, a slight decrease in tendency to attack is observed (see section 5.5.2). When positive reinforcement arrives, UE+ is set to 600, and thereafter it decays as $300(1-log(1+0.002T))$, where T is the number of cycles since reinforcement. When negatively reinforced, UE- is set to 600 and decays as $300(1-log(1+0.002T))$. The resting values for both UE+ and UE- are reached after 900 cycles, or 1.5 hours of simulated time.

5.3.7. Simplifications in OVSIM

Several effects are observed in *Octopus* which allow OVSIM to be simplified without too much deviation.

- The octopus will eat up to 20 10-15g crabs per day [BoY55].
- During experimentation, octopuses are maintained in a constant state of hunger (e.g., [You60]).
- Once trained, the probability of the octopus attacking an object is independent of hunger [You58b].
- Octopuses trained not to attack the negative figure, and then starved, would not attack that figure even if it was shown three times a day for three days [BoY55].

Therefore, no hunger, satiation or diurnal effects are incorporated into the OVSIM model.

- During training, the animal is usually to be found in its 'home' at one end of the tank, and the training stimuli can be presented at the other end of the tank, a near constant distance of 80-90cm [BoY55]. Therefore training occurs at one retinal size.
- Octopuses have not been shown to generalise over size changes [You64, p.172].

The MPLN networks used as the adaptive memory in OVSIM have no inherent size generalisation capability, and this issue has not been considered further.

- The octopus usually watches as the stimulus figure is lowered into the tank, and then, if the figure is positive, attacks as soon as the figure touches the bottom of the tank.

OVSIM trials begin with the stimulus pattern centered on the retina, and do not con-

sider the lowering into view of the stimulus or the tracking mechanisms by which the octopus locates the stimulus.

There are several further effects in *Octopus* which are ignored in OVSIM because a decision was made to restrict OVSIM to modelling the visual attack learning system only:

- Binocularity and transfer of information between the optic lobe associated with each eye.
- Interplay of other simultaneous systems involving camouflage, mating, etc.
- The touch learning system which is quite comparable to the visual one.
- Ability to operate in continuous time with much more complex retinal information.
- Existence of the antagonist retreat learning system.
- Simple forgetting, as opposed to forgetting induced by interference from new memories.

Hopefully, some of these issues may be addressed in future work.

5.4. Trials with the Simulated *Octopus*

Using OVSIM it is possible to perform a number of learning tasks which are made to model, as closely as possible, experiments with *Octopus*. Some comparisons are perforce limited: for example, it is difficult to calculate and replicate within a computer program all of the subtleties of an octopus learning to reject a proffered crab, its natural food source. Still, repeatedly, the OVSIM learning curves share many characteristics with learning curves obtained for *Octopus*. This suggests that OVSIM has some success in "surviving" in an environment like that which *Octopus* faces, and also that the ADB model may share some facets with the learning system in *Octopus vulgaris*.

5.4.1. Discrimination tasks

One task which the octopus learns fairly quickly is discrimination of vertical and horizontal rectangular figures – attacking one and rejecting the other. Wells describes animals which can learn this task within about 10 trials [Wel68, p.168]. The animals can also easily learn to attack a crab (their natural food source) when shown alone, but to reject it when paired with a white plastic square; Boycott and Young found that the octopuses could learn this discrimination within about 10 days at three trials a day [BoY55]. This task not only involves learning not to attack a food object, but also involves a negative stimulus which actually subsumes the positive stimulus.

OVSIM was trained first to discriminate between patterns of vertical (positive) and horizontal (negative) rectangles, p_V and p_H[7]. In these experiments, $\hat{s}$ was set to 0.35, and so the system had a probability of 0.2 to attack a random figure – this was chosen to match with the data from an octopus experiment ([You64, p.226]). With that initial setting, V, the number of MCs voting to attack, approached unity to p_V and zero for p_H. The evolution of V is shown in Figure 5.4. Figure 5.4 also shows the evolution of percent attacks by *Octopus* in the vertical/horizontal discrimination task; the two curves have close fits in shape.

OVSIM could also learn to accept the crab image p_C and reject the crab-plus-square p2C within about 10 trials. This task does not retain the biological significance of the *Octopus* task, but shows that OVSIM can learn when the negative stimulus subsumes the positive one.

[7] All OVSIM stimuli in the experiments reported here are shown in Appendix B and referred to by the abbreviated names given them there.

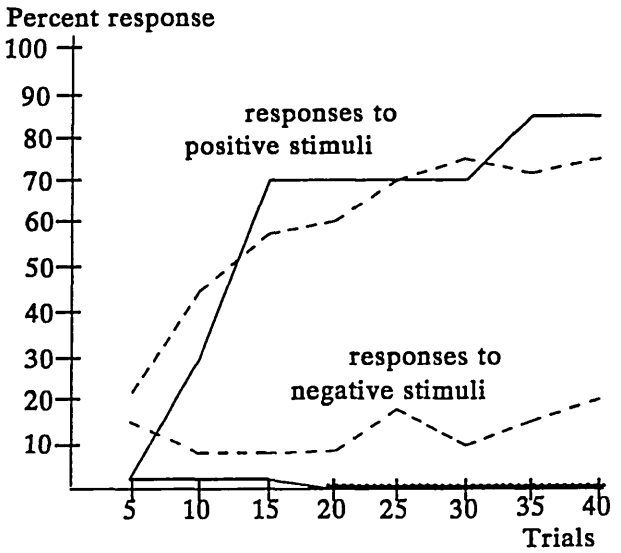


Figure 5.4. Learning discrimination tasks. The solid line shows the evolution of V (as percentage of MCs voting to attack) in OVSIM to the positive p_V and negative p_H patterns, averaged over three simulations. The dashed line shows the percent attacks by Octopus *as a function of trials for the same task (from [You64, p.226]); data averaged from 25 animals.*

5.4.2. Fall in delay to attack

When the experimenter trains an octopus to attack crabs (or any stimuli) in the tank, the animal at first attacks only after a long delay or observation period which may take several minutes [You64, p. 71]. With repeated trials, the octopus gradually decreases both this delay and the overall time taken to carry out the attack. Eventually, as shown in Figure 5.5a, the octopus will attack as soon as the stimulus touches the tank floor.

Learning in OVSIM shows the same characteristic decrease in delay to attack; Figure 5.5b illustrates the effect for learning to attack the crab pattern p_C. Initially, when the number V of MCs voting for attack is low, it will on average take several cycles before a system attack occurs; as V rises, the average waiting time to attack falls until attacks will usually occur on the first cycle of each trial.

The inverse situation occurs to negative stimuli in both OVSIM and *Octopus*: delay to attack a negative stimulus gradually increases, until attacks cease altogether

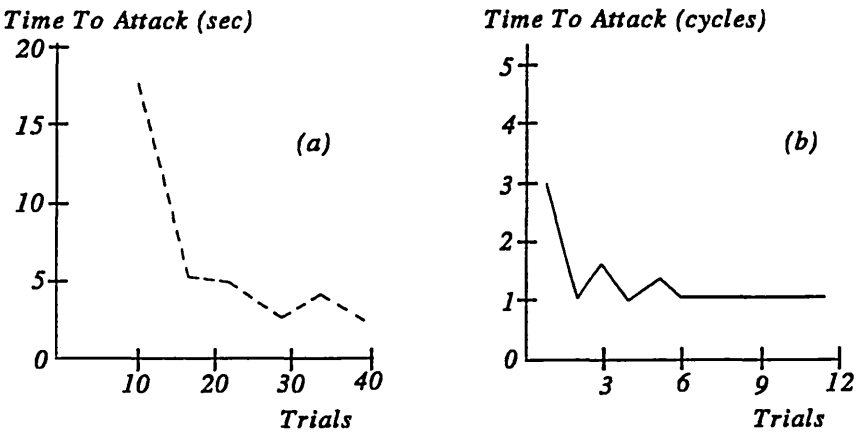


Figure 5.5. The fall in delay to attack a positive stimulus (crab) in (a) Octopus, *averaged over 10 animals, and in (b) OVSIM, averaged over 3 simulations.*

within the finite length of a trial. Delays never cease altogether in *Octopus* [MuG88], and if the negative stimulus is left in the visual field indefinitely, an attack will occur eventually. In OVSIM, attack is probabilistic, and given enough time an attack will also occur as long as the probability is non-zero.

The occurrence of this phenomenon in OVSIM as a result of its probabilistic nature lends support to the view that the *Octopus* may also learn by adjusting probabilities of attack, and repeatedly polling them throughout a trial.

5.4.3. Relearning

Another learning characteristic which OVSIM shares with *Octopus* is that recent information may disrupt previously stored information, although the old information is seldom lost entirely.

Boycott and Young [BoY55] subjected several animals to a relearning task. Each octopus was given three trials with a crab, which it was allowed to eat, followed by three trials with crab and square and shock, if attacked. This routine was repeated for several days. Each day, the response to the negative stimulus decreased; but overnight and after interference from the positive trials, the response to the negative stimulus would have regained some strength, and rejection would

have to be relearned. However, the amount of relearning required each day decreased, implying that long-term memories were being formed.

This experiment was simulated in OVSIM using the crab (p_C) and crab-plus-square (p2C) patterns as positive and negative stimuli. The learning routine consisted of alternate blocks of three positive and five negative trials. Figure 5.6 shows the change in probability to attack each stimulus. The tendency to attack p_C grew steadily; after trials with p2C, the response to p_C dropped and had to be relearned.

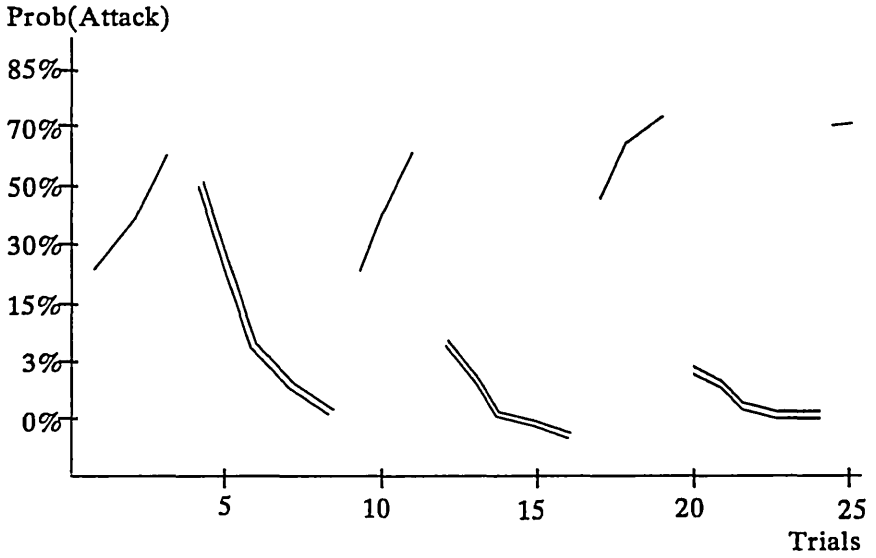


Figure 5.6. Learning with interspersed blocks of positive (p_C) and negative (p2C) trials; solid line is trials with p_C, double line is trials with p2C. Each block of training interferes and weakens the opposite memory; but as learning proceeds the interference is less. Data averaged from 10 simulations.

Each block of trials with p_C begins with a slightly higher response than the previous block, until by the fourth block of trials, there is little loss of learning. Meanwhile, the situation is reversed for the negative stimulus: the response to p2C steadily decreases within each trial, but rises for the beginning of each block of negative trials. By the final block of negative trials, there is little disruption of the response.

OVSIM's memory is such that any positive reinforcement increases the probability of attacking an arbitrary figure. Therefore, in the absence of negative rein-

forcement (as during a block of positive trials), the response to the negative stimulus also rises. As the negative memory is strengthened over repeated trials, the effect of the positive reinforcement is less able to interfere. Similarly, as the positive memory is strengthened, negative reinforcements disrupts it less. Because this effect mirrors that in *Octopus*, it is again support for a view of *Octopus* as a probabilistic machine with characteristics similar to those in OVSIM.

5.4.4. Transfer of discrimination learning

Octopuses which have learned to attack a crab but not one shown with a white square show some transfer of this learning to similar situations: they will not attack a crab shown with a circle or triangle of equal area, or one shown with a square of half the area of the original [BoY55]. This indicates that the optic lobe encodes these patterns more or less equivalently.

In the case of the OVSIM simulation, it is possible to determine exactly how the CCs classify input, and thus to predict how the system should respond to new input.

15 OVSIMs were trained to make the crab (p_C) versus crab-plus-square (p2C) distinction. The square was then replaced in test patterns by a smaller square (p3C), the outline of a square (pFC), a circle (pcC), a reversed circle (pRC), and a triangle (pTC). Figure 5.7 (top) shows the percent overlap of each new composite picture with the original positive and negative stimuli – in terms of how many CC outputs change. (Each instantiation of OVSIM has identical CC response to a pattern). From this measure, it is to be expected that the system would respond to pcC in a strongly negative fashion, to pTC and pFC in a more weakly negative fashion, and to p3C in a strongly positive fashion. Figure 5.7 (bottom) shows that the average actual responses to each pattern are consistent with these predictions. A response near 800 votes – half of the MUs active – would be noncommittal.

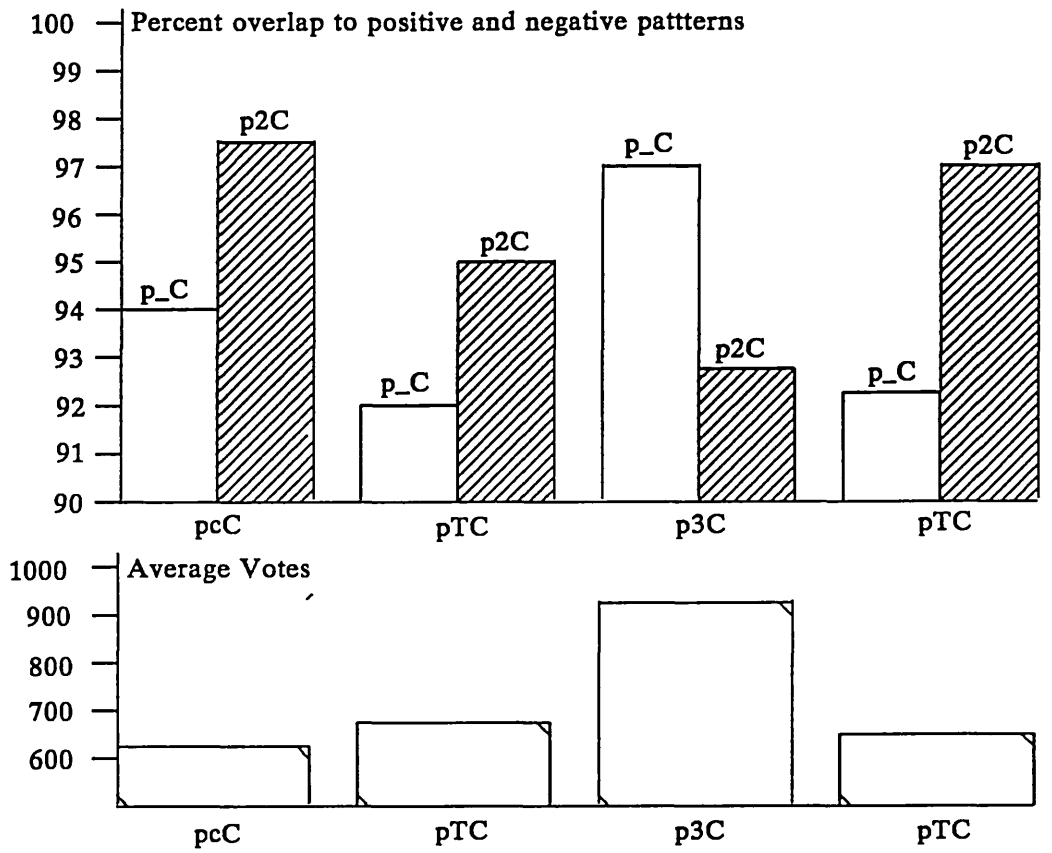


Figure 5.7. Transfer of learning. The percent overlap (top) of new patterns with the positive (p_C) and negative (p2C) trained patterns suggests tendency to attack should be strongest for p3C and weakest for pcC; actual system responses (bottom) agree. Response data is average over 15 simulations.

One indication of these experiments is that OVSIM encodes retinal input in at least partially the same way as does the octopus optic lobe. The situation in the animal is obviously more complex than in the simulation. By experimenting with different combinations of classifying cell type and number in the simulation, it should be possible to obtain responses which mimic the octopus's closely, and which therefore provide some constructive hypotheses about how the animal accomplishes this task. Unfortunately, such experimentation probably requires much closer correlation between visual input to OVSIM and to *Octopus* than is feasible without more sophisticated visual input capability in the simulation.

5.4.5. Multiple discrimination task

Boycott and Young [BoY56] have shown that octopuses can hold at least three discriminations in memory at once. They trained animals to attack a small square but not a large one; these trials were then interspersed with training to attack a vertical and not a horizontal rectangle; and finally, the animals learned also to accept a white circle and reject a black one. Figure 5.8 shows how training progressed for one animal.

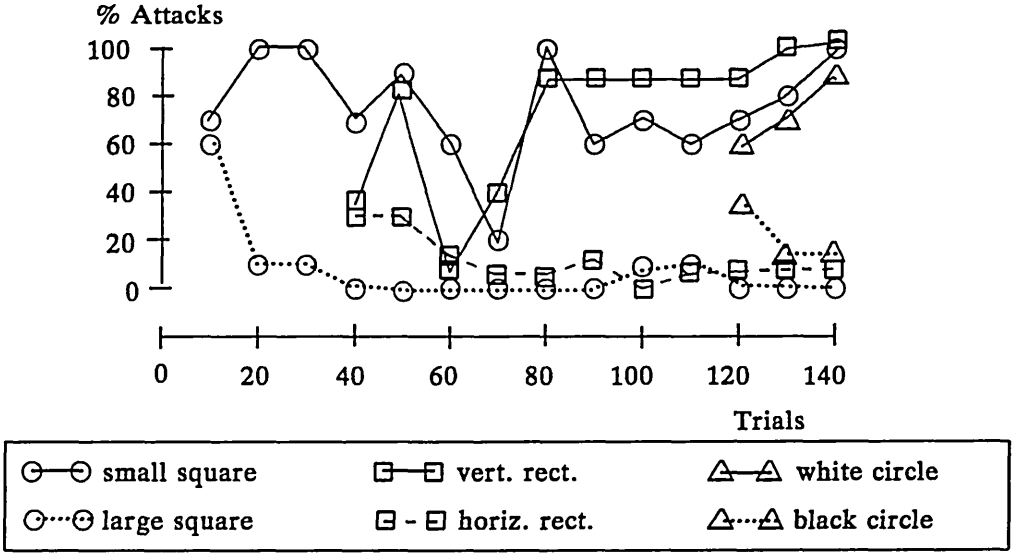


Figure 5.8. Multiple discrimination learning in Octopus, *adapted from [You64, p. 132].*

The following points may be noted:

A. The second, vertical/horizontal discrimination takes some 40 trials to learn. Ordinarily, this is an easy discrimination for the octopus to learn – and should perhaps take only 10-20 trials to learn, as shown in Figure 5.5.

B. When the second discrimination is introduced, response to the first stimulus pair becomes less accurate. Errors are mostly failures to attack positive stimuli.

C. After introduction of the second stimulus, the first discrimination never quite regains its previous accuracy.

D. Adding the third, white/black discrimination task interferes little with the previous two. Young [You64 p. 133] hypothesises that this is because contrast discrimination is recorded by a different set of cells from those encoding shape discrimination.

E. Eventually, all six stimuli elicit mainly correct responses.

OVSIMs were trained to attack a 4x4-bit square (p_3) but reject a larger one (p_1) until three responses in a row to each were correct. Trials with these patterns were then interspersed with learning to attack a vertical bar (p_V) but reject a horizontal one (p_H) and finally with learning to attack a circle (p_c) but reject its inverse (p_R).[8]

Figure 5.9 shows learning behaviour averaged over six OVSIMs.

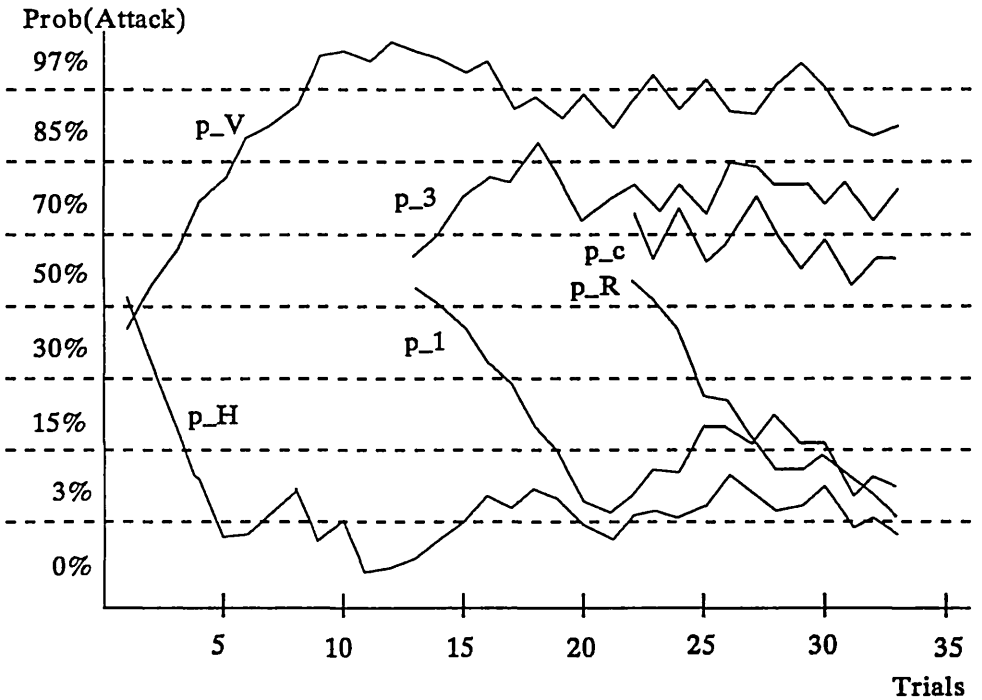


Figure 5.9. Multiple discrimination learning in OVSIM; each point is average response from 5 simulations.

Comparing with Figure 5.8 several consistencies appear which address points A-E

[8] Patterns p_3, p_c and p_R are derived from pCc and pCR shown in Appendix B, but with blank 16x16 fields replacing the crab images.

above:

A'. The time to learn the second, p_V/p_H discrimination averages 10.00 attacks or 11.00 trials for the six experiments. Three naive systems learned to make this discrimination significantly faster: within an average 6.67 attacks or 7.67 trials. Both of these differences are significant by t-test ($P<0.025$).

B'. Introduction of the second discrimination to OVSIM does disrupt the previous knowledge, as shown in Figure 5.9 by slight returns to randomness in response to both p_3 and p_1 when p_V and p_H are introduced. The average response to p_3 falls from 1128 votes to 1075.67 votes ($P<0.025$ by t-test), and that to p_3 jumps from 539 to 566 votes ($P<0.10$ by t-test). Both are definite trends – the fall in positive response is particularly significant.

C'. In *Octopus*, the response to the first discrimination never regains its former accuracy once the second discrimination is introduced. However, in simulations, the level of responses were not found to differ significantly. As neither loss of accuracy was significant in the simulation, it suggests either that the simulation memories were stronger or suffered less interference than did the octopus memories. Perhaps the octopus memory would also have been regained with a longer retraining time.

D'. Addition of the third (black circle versus white circle) discrimination interferes little with the previous memories in *Octopus*. In OVSIM, there are slight disruptions to all memories, but only that to p_H is significant by t-test – the response drops from 599 to 630 votes ($P<0.01$). Young [You64 p.133] claims that in *Octopus* the disruption caused by introduction of the vertical/horizontal discrimination shows that representations for the four figures in the optic lobe overlap; the black/white distinction is presumably encoded in a different set of cells and so the memories overlap less. In OVSIM, it is more likely that earlier memories may be well fixed by the time of introduction of the third

stimulus pair and so are not noticeably disrupted. If this is also true in *Octopus*, this would also explain why successive discriminations are learned with successively less strength – less unfixed memory is available to encode them.

E'. Eventually, in OVSIM as in *Octopus*, all learned responses are basically corrected. At the end of the simulation, each positive stimulus was attacked with a probability of at least 70% on any given cycle, while probability of attacking for each negative stimulus was less than 3%.

In summary, then, all effects seen in *Octopus* are quite well matched in OVSIM, with the exception of C. which would predict more disruption to the first discrimination. The simplest explanation for this would be that the animals just required more training time to regain the first discrimination.

5.5. Damage Experiments with *Octopus* and OVSIM

Showing that OVSIM bears behavioural resemblance to *Octopus* is showing performance equivalence between a computer program and a real animal. Section 5.4 has made some claim to this. It went further to suggest that OVSIM might provide a partial model of the animal: in particular, that it might be appropriate to view *Octopus* as a machine which learns by adjusting the probabilities of selecting various actions under various conditions, according to their expected results.

In this section, it is shown that, when various parts of OVSIM are removed or damaged, the simulation loses function in manners quite analogous to loss of function after surgical ablations in *Octopus*.

5.5.1. Damage to the optic lobe

The optic lobe in *Octopus* is believed to contain classifying cells, memory cells, and a region which generates output containing the decisions to attack [Mal63]. In OVSIM, these correspond to CCs, MCs and the output generator, respectively (see Section 5.3). To simulate ablation of the optic lobe, OVSIM is damaged such that some fraction of its MCs output noise (random 0/1) under any input. The output-generating and attention-assigning modules are maintained intact.

Boycott and Young experimented with octopuses after bilateral removal of about 50% of optic nerve fibres or or optic lobe tissue (which should be equivalent losses). They found that the animals were capable of learning discriminations "in the usual way", presumably at normal speed and to normal strength [BoY55].

In OVSIM, this damage was simulated by randomly selecting 50% of the MCs and forcing them to only output noise (random 1s and 0s). Three OVSIMs in which this was done were able to learn to discriminate the vertical and horizontal bar patterns pVV and pHH. Table 5.1 compares the results on the damaged OVSIMs with those of normal ones; the responses learned do not differ significantly in strength, but the damaged systems take considerably longer to learn. This increase in learning time does not apparently occur in damaged *Octopus*. In OVSIM, the result of learning with less available memory is longer time required to find a suitable representation among the remaining cells.

	Time to Converge	Avg Votes (+ve)	Avg Votes (-ve)
Normals	8.67	1077	538
Damage, then train	31.0	1021	586
Prob (from t-test)	$P<0.005$	$P<0.25$	$P<0.05$

Table 5.1.: Time to converge and average response (over 5 trials) to the positive and negative stimuli of 4 OVSIMs without and with damage to MCs.

Boycott and Young [BoY55] also tried the reverse experiment: they removed 50% of the optic lobe from an octopus already trained to discriminate two figures,

and found that the animal's performance degraded little. They also removed between 25-50% from three other trained animals; one did not attack again, and the others retained the discrimination well.

An OVSIM was trained to discriminate pVV and pHH; after 10 trials, it attacked the positive stimulus 10 out of 10 times and the negative 2 of 10 times. At this point the simulation was "damaged" so that half of the MCs would only output noise. Because this was done in simulation, several random lesions could be compared on the same trained OVSIM. Each gave very similar results, as shown in Table 5.2.

OVSIM	% Damage	Before (+ve)	After (+ve)	Before (-ve)	After (-ve)
1	50%	1159	995.8	515.2	670.4
2	50%	1159	999	515.2	670.4
3	50%	1159	1012	515.2	666

Table 5.2.: Average of five responses in votes by 3 damaged OVSIMs to the positive and negative stimuli.

The response to the positive stimulus dropped markedly by an average 13.5%, while the average response to the negative rose by about 20.9%. However, the simulations could still consistently discriminate correctly, and would attack the positive stimulus with 70% probability on any one cycle, and the negative stimulus with probability of only 15%.

5.5.2. The Unspecific Effect and vertical lobe damage

After feeding (even in the home) the octopus is more likely to attack any arbitrary stimulus, while after shock it is less likely. This effect declines over 1-2 hours [You58b] until the octopus attacks with usual frequency. Maldonado has named this the Unspecific Effect [Mal63] (as opposed to a specific effect of a shock to reduce further attacks to that particular figure).

OVSIM mimics this effect to a slight degree implicitly. When the simulation's memory is reinforced positively, one or more locations in each MPLN are

incremented. The probability that an I-input MPLN will output 1 in response to an arbitrary pattern is given as:

$$Prob(output=1)=\sum_{j=1}^{2^I} Prob(j\ addressed)*sv_j=\frac{1}{2^I}\sum_{j=1}^{2^I} sv_j \qquad 5.3.$$

where sv_j is the value stored at location j. Therefore, even one positive reinforcement increases a node's probability of outputting a 1 (and hence contributing a vote) to an arbitrary pattern. A negative reinforcement decreases this probability. On a system-wide level, this leads to a small Unspecific Effect. It is this which causes the "unlearning" effect shown in Figure 5.6.

OVSIM also incorporates explicit factors by which the Unspecific Effect is simulated. Maldonado [Mal63] suggests that there is a quantity, the Unspecific Effect Parameter (UEP), by which the decision to attack is amplified or suppressed. In OVSIM, this is subdivided into UE+ and UE-, as explained in Section 5.3.6. The total number of votes plus UE+ minus UE- is then used in the generation of a probability to attack. Maldonado lists some support for the idea that there are two separate functions in *Octopus* – one each to implement the negative and positive signals [Mal63].

In the OVSIM results mentioned so far, at least 900 cycles (1.5 hours simulated time) has been allowed between trials, and so the explicit Unspecific Effect has not been relevant. Octopus trials are often similarly spaced by at least 40 minutes to avoid confounding effects of learning with Unspecific Effects (e.g., [You58a]).

If, however, OVSIM trials are separated by less time, explicit Unspecific Effects appear. If an OVSIM is given positive reinforcement for attacking p_C and then responses to a new figure p_H are tested at intervals of 150 cycles (15 minutes simulated time), there is at first a dramatic drop in the delay to attack, caused by the rise in the quantity $V+$(UE+)-(UE-). This decays with time until by 900 cycles (1.5 hours) the response to the neutral figure is back to its normal levels. Figure 5.10a

shows the delay to attack, averaged over 6 OVSIMs, together with UE+, for the interval.

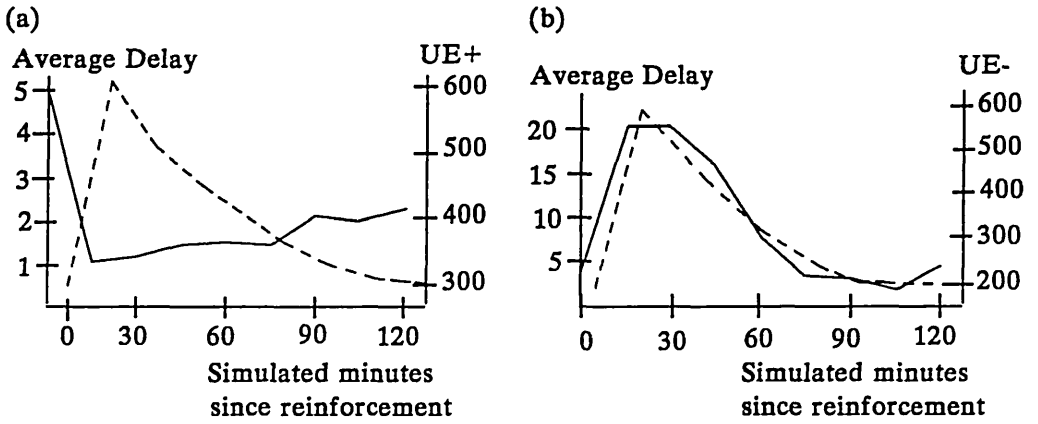


Figure 5.10. Effect of UE on delay to attack. (a) Solid line shows delay to attack in cycles after positive reinforcement at time 0; dashed line shows evolution of UE+. (b) Solid line shows delay to attack in cycles after negative reinforcement at time 0; dashed line shows evolution of UE-. Both figures show data averaged from 6 OVSIMs. 1 minute simulated time = 10 cycles.

The same situation, but preceded by negative reinforcement for attacking p2C, is shown in Figure 5.10b. It shows a dramatic increase in the delay to attack the neutral figure, which eventually returns to normal levels.

When octopuses which have been trained to some discrimination and then had vertical lobe ablation are retested, they show a significant increase in attacks to the negative pattern, suggesting that the vertical lobe in some way produces or manipulates UE- [You64, p.226-229]. When the lateral superior frontal lobe is removed, the animals show a decrease in attacks to the positive figure [You64, p.230], so the lateral superior frontal seems to be involved with adjusting the UE+. If both vertical and lateral superior frontal lobes are ablated, the animals attack all patterns slightly less than usual [You64, p.228]. This implies that, at rest, UE+ and UE- should produce a net positive effect, and this is why resting values of UE+ at 300 and UE- at 200, for a sum of UE+ - UE- = 100 > 0, are defined in Section 5.3.9.

In an OVSIM trained on the crab (p_C) versus crab-plus-square (p2C) task, fixing UE- to 0 results in a system which attacked all stimuli, even the ones it had

previously learned were negative. Fixing UE+ to 0 resulted in slightly longer delays to attack the positive figure than normal: averaging 3.0 cycles to attack instead of 1.0. If both UE+ and UE- are eliminated, the system performs quite well, with only a slight increase in the delay to attack positive figures: averaging 1.2 cycles delay. The effects of removal of the UE+ and UE- therefore share at least these effects with removal of the superior frontal and vertical lobes, respectively.

5.5.3. Simulating vertical lobe ablation

When the UE- mechanism and the ADB buffer in OVSIM are damaged together, the resulting effects are quite parallel to effect arising from vertical lobe removal in *Octopus*. This is accomplished by (1) the loss of UE- by setting UE-= 0; (2) destruction of the attention-assigning mechanism by setting it to output a constant (meaning that buffering is recency-driven rather than attention-driven); (3) restriction of the buffers to hold a single element – which by (2) will always be simply the preceding input. This, like vertical lobe ablation, can be done before any training takes place (R-T paradigm) or training can be followed by removal and then further training (T-R-T paradigm).

An R-T octopus will be unable to learn to avoid attacking crabs [You70], typically stopping after a shock but trying again 1-2 trials later. It will also be unable to learn discriminations involving crabs – such as crab positive unless shown with square [BoY55].

However, some learning about neutral stimuli by these animals has been documented in [Boy67] and [You65b]: in one experiment, after 180 trials, the animals would attack the positive stimulus five times as often as the negative [You58a].

Ordinarily, an OVSIM can discriminate crab (p_C) versus crab-plus-square (p2C) within some 10-15 trials (Section 5.4.1). R-T OVSIMs were still regularly

attacking p2C by the 25th trial. Pattern p_C, unsurprisingly, was consistently attacked. In the long term, the R-T OVSIMs did show some learning: a decrease from 100% attacks on p2C in the first five trials to 80% attacks by trials 20-25. The response in votes to p2C had dropped to an average 477 in trials 20-25 (whereas p_C was eliciting some 917 votes), but this was still too strong a response to counteract UE+ in the absence of UE-.

After training a T-R-T octopus to attack crabs unless paired with a square, and then ablating all vertical lobe tissue, Boycott and Young found that the memory preventing attack was lost and could not be relearned [BoY55]. If at least 30% of the lobe was intact, the animal behaved normally; with larger lesions, behaviour was degraded until by 80% removal the animal was unable to learn. They found no evidence that the memories lost in these animals with major lesions could ever be reacquired.

On the other hand, discriminations involving neutral stimuli could usually be recovered "quite well" [BoY56]. Typically, discrimination memories lose some accuracy after vertical lobe removal but slow relearning is possible. It also appears that disruption is worse for memories associated with difficult discriminations. Possibly, these memories are weaker even before operation, and therefore less able to withstand the loss of UE- [BoY56].

Four T-R-T OVSIMs were trained to the p_C versus p2C task; after damage, all but one OVSIM attacked every pattern presented. Even after 20 cycles of training there was no improvement. Much of this failure is attributable to overwhelming probability of attacking any pattern in the absence of UE-.

However, comparing the response in votes to positive and negative patterns in these simulations with those from the R-T simulations shows that some memory exists in the T-R-T set. The average response to a positive figure in T-R-T is 989.70 votes (std. dev. 9.539), while it is only 955 votes in the R-T set. For

negative figures, the average T-R-T response is 453.80 votes (std. dev. 7.097) compared with 527.3 in the R-T set. In both the positive and negative conditions there is a highly significant savings of the previously learned information ($P<0.005$ from t-test in each case).

5.5.4. Overcoming loss of UE- and the ADB buffer

Boycott and Young found that, in octopuses with vertical lobe ablation, memory would be prolonged if the negative crab-plus-square stimulus was left moving in the visual field after the shock was given [BoY55]. They suggest that normally the vertical lobe system somehow performs this re-stimulation of the optic lobe cells – accomplishing what the experimenters did by hand – so that the stimulus image is still available when reinforcements arrive [BoY55].

In normal OVSIM, when reinforcement arrives, every item in the buffer is reinforced for $H-10$ cycles. If the attention-assigning mechanism is damaged and if the buffer size is one, then the inputs being reinforced will be the H patterns immediately following reinforcement. If trials are separated by the usual 900 cycles, then most reinforcement will occur to blank patterns. If, however, the next trial follows immediately after the last, then the new patterns will receive reinforcement; in a rejection task, the re-presentation of the negative item will therefore enter the buffer in time to receive reinforcement from the *last* attack. As a result, the system will learn exactly the correct association despite its damage.

Figure 5.11 shows rejection learning averaged over three damaged OVSIMs. When 900 cycles (1.5 hours simulated time) separate trials, there is some learning, but the response stabilises at about 460 votes. If the trials are not separated, the response can drop low enough that the probability of attack is effectively zero. (Initially, the systems learning with separated trials learn faster. This is simply because, in the continuous trial condition, a new attack may occur on the very next cycle, and

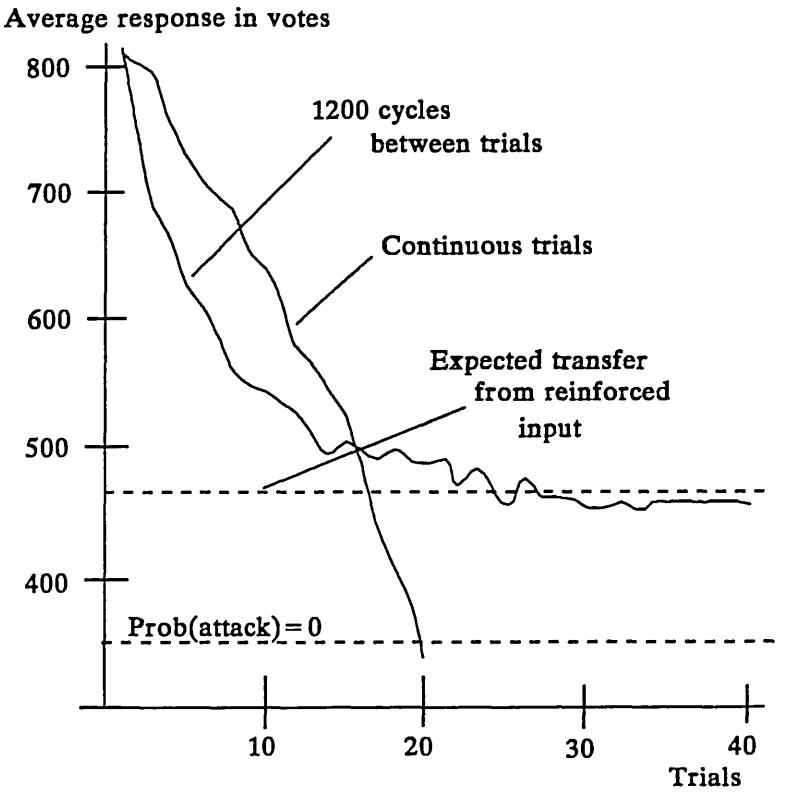


Figure 5.11. Average response in votes of damaged OVSIM learning rejection of p2C, when trials are separated by 900 cycles or 1 cycle (continuous trials). With continuous training, probability of attack falls to 0. With separated trials, the system learns to reject the pattern occurring when reinforcement arrives, and the response to stimulus is merely generalisation from that.

the H cycles of reinforcement will be interrupted.)

The level at which the response with separated trials stabilises is actually artificially strong. The input present when a reinforcement arrives is actually the fourth one after the original stimulus appears: as shown in the sequence in Appendix C. With damage, that is the input which will always be in the buffer to receive reinforcement, and it is therefore the only pattern reinforced. The overlap between this and the original stimulus is some 89.095% of CC outputs, and so the number of MCs expected to receive identical inputs for both patterns is the number expected to receive all inputs from within that overlap:

$$686(.89095)^6 + 515(.89095)^8 + 414(.89095)^{10} = 677 \text{ (}of\text{ 1615)} \qquad 5.4$$

These will output 0 for both patterns. It is to be expected that, of the remaining

MCs, one-half will output 0 and one-half 1. Therefore, the number of MCs expected to output 1 to the true stimulus pattern is:

$$(1-0.4192)(0.5)(1615) = 469 \qquad 5.5$$

The actual average number of MCs responding to the stimulus pattern by trials 36-40 in the 1.5 hour case was 455.

Therefore, all "learning" about the stimulus in the separation trials is artifact – transfer from learning about the picture visible when reinforcement arrives – and no real delay learning takes place.

On the other hand, when the separation between trials is only one cycle, the OVSIMs can learn rejection in about 20 trials – only a few more trials than normal OVSIMs.

5.6. Conclusions and Summary of Chapter 5

OVSIM, the ADB-based model, has been demonstrated to be capable of learning delayed reinforcement tasks modelled on those learned by *Octopus vulgaris*. These are primarily discrimination tasks where reinforcement follows grasping and drawing in of the stimulus object – so that the visual input changes before the reinforcement arrives.

OVSIM shows many of the same characteristics of learning as the octopus: most notably the changes in delay to attack with learning. These effects arise in OVSIM because of the manipulation of a probability to attack which is repeatedly polled over the length of a trial. Perhaps this is like what goes on in *Octopus*. The alternative hypothesis is that a delay is due to a slow building up of activity to cross some threshold required for attack, and with learning the activity starts at a higher level and thus crosses the threshold faster [Mal63].

Vertical lobe ablation in *Octopus* was shown to share features with the simul-

taneous elimination of UE- and of the attention-assigning mechanism in OVSIM. If attention is always set to a constant and if the buffer size is one, the buffer only maintains the single previous input, and this leads to deficits like those in *Octopus* without vertical lobe. While there is no real evidence that the octopus system operates after an ADB-like fashion, at least the data about vertical lobe removal in *Octopus vulgaris* seem consistent with ADB removal in OVSIM.

5.6.1. Potential improvements to OVSIM

There are several ways in which OVSIM could be improved. Most obviously, its inputs are highly stylised. A more advanced implementation might use video input for its stimuli, and also during the delay until reinforcement; and it would then have to reconcile "continuous time" input with buffers which can only hold discrete images.

In the octopus, output from the optic lobe seems to emerge via two competing pathways – one carrying the decision to attack, one the decision to retreat. Conflict between them would need to be resolved either through lateral inhibition or top-down (goal-directed?) control. Little enough is known about this process in *Octopus* that it seemed a reasonable phenomenon to omit in OVSIM. But it would be interesting to provide OVSIM with a second retreat-learning subsystem and study the interactions of the two systems during learning.

Octopus also has capacity for transfer, so that associations learned using one eye (and hence one optic lobe) and tested in the opposite eye are retained. This would be simple enough to achieve in OVSIM, by providing a second set of CCs and MCs as the second optic lobe, and then wiring corresponding MCs together and letting them share values. This does not seem a likely solution in *Octopus*: it would require links between 120 million nerve cell pairs, each of whose function is surely subject to change during development. Yet this is essentially the hypothesis put

forward by Young [You64, p.170] and Maldonado [Mal63] concerning the ventral commissure joining the optic lobes. A different solution might be for a sampling of retinal cell axons to travel to the ipsilateral optic lobe, instead of contralateral as usual. This would provide each optic lobe with (slightly less detailed) intraocular information, at the pre-CC stage. Then no explicit transfer would be needed, as MCs from both optic lobes would learn in the usual way, no matter which eye was providing the input. Unfortunately, Young reports that "the chiasma is certainly almost if not quite complete" [You71, p. 449]. Octopus experiments on differential learning with severed ventral commissure versus severed subvertical to optic lobe connections might show from where the contralateral optic lobe was actually receiving its signal to learn.

OVSIM CCs have been constructed to match the stimuli classes to which the octopus responds well, under the assumption that vertical/horizontal distinctions, for example, are easy because cells explicitly encode this feature. There is, in fact, very little evidence about what types of cells actually exist; suggestions range from simple vertical/horizontal and black/white recognisers to the very complex ones posited by Sutherland (ratios of maximum vertical and horizontal extent to square root of area [Sut57] or openness/closedness of figures [Sut63]). This is all deduction from observed behaviour, as there seems to have been no single cell recording from *Octopus* optic lobe. It might be interesting to use OVSIM to "reverse engineer" classifying cells: use some strategy, such as a genetic algorithm, to generate CCs which resulted in OVSIM behaviour closely matching that of *Octopus*, and then inspect the CCs so derived. This would of course require a much more sophisticated OVSIM than the ones used here.

5.6.2. Maldonado's model of *Octopus*

It is appropriate to describe here the model of *Octopus* visual attack learning which was developed by Maldonado [Mal63], and which provided a great deal of the groundwork for OVSIM: at least in terms of dividing the optic lobe explicitly into classifying and memory cells, in explicit provision of a UEP, and in treating the attack/retreat decision as a simple scalar.

Maldonado divided the optic lobe neurons into three types: classifying units (CUs), memory units (MUs), and an addition unit (see Figure 5.12). Taken together, they function very much like a single-layer perceptron. The CUs receive input from a localised region of the retina, and classify it according to the presence or absence of some feature such as brightness, shape or movement. For each possible output of each CU there exists an MU, which is "sensitised" by receiving CU input.

Visual memory is stored in MU response levels (much like perceptron weights), which are adjusted upon receipt of various signals, particularly from the noci-hedono receptors. MUs are only receptive to this input when sensitised by their CU. The MU weights decay in the absence of reinforcement. It is possible that MUs may differ from one another in the range of values which their weights may assume.

In the addition unit, the responses of all MUs are transformed into a weighted sum which represents the strength of the decision to attack. Conversely, a weak signal indicates retreat. These two actions are the only two behaviours considered.

The attack/retreat decision leaves the optic lobe and travels to the supraoesophageal regions, which Maldonado perceives as a two-stage amplification system: the medial superior frontal lobe amplifies the positive signals, and the vertical lobe amplifies the negative ones. The resultant output, the Experiential Parameter (EP),

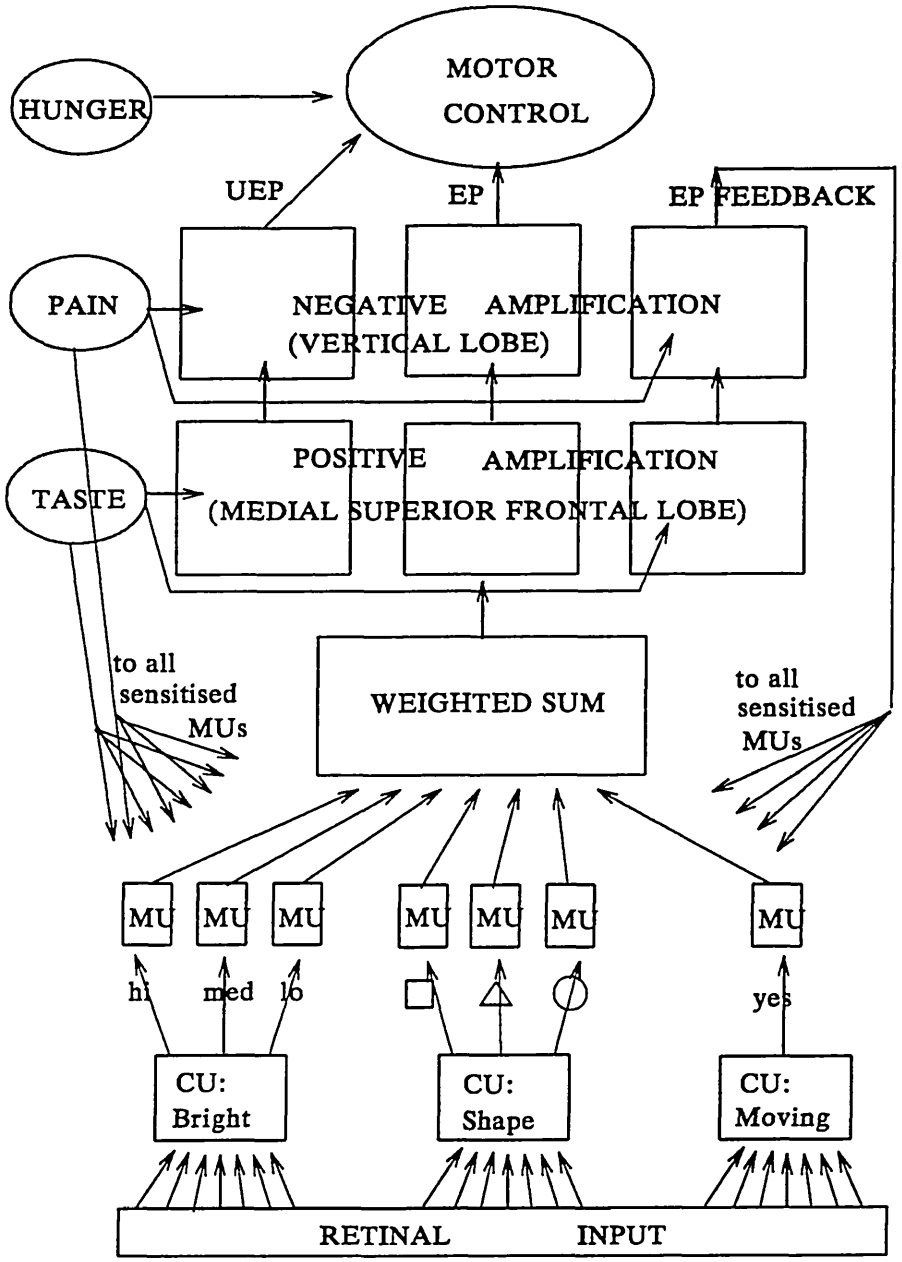


Figure 5.12. Maldonado's model (adapted from several figures in [Mal63]). Retinal input drives CUs which signal the presence or absence of features such as brightness, shape and movement; their outputs drive adaptive MUs – one for each possible output of each CU. The summed MU responses pass out of the optic lobe as a strength of decision to attack which is amplified positively by recent taste and negatively by recent pain. Feedback of this response maintains active (firing) MUs at low threshold; when reinforcement arrives, all MUs at low threshold are updated.

passes to a command centre which in turn feeds into the higher motor regions. Maldonado's EP is supplemented by two other, non-learning-related parameters: hunger, which always serves to make attack more likely, and an Unspecific Effect Parameter (UEP). The UEP amplifies signals from the nocihedono receptors, so that after pain is received, the octopus shows an overall reluctance to attack arbitrary stimuli, while after eating it becomes more likely to make further attacks. Again, the medial superior frontal amplifies positive UEP signals while the vertical lobe amplifies negative ones. These also pass to the motor system, so the launching of an attack is a function of the three: EP, hunger (to a lesser extent) and UEP.

Some fraction of the EP is fed back to all sensitised MUs, making them slightly more likely to repeat the last decision. This provides for the observed effect that an octopus's initial delay in attacking a novel stimulus gradually fades with training, as EP builds up in the cells. It also provides a sort of short-term memory, allowing the persistence of optic lobe representations until reinforcement arrives. Although an animal whose vertical lobe has been ablated cannot learn to reject a stimulus associated with shock, it performs better when the stimulus is left in the visual field [BoY55], implying that retinal input may keep the MUs sensitised in the absence of an EP to accomplish the same task. Nocihedono input, when it arrives, is applied to all MUs which are still sensitised.

Much of this compares with OVSIM. It differs in the following important points:

- Each of Maldonado's MUs is associated with a single output from a single CU. This seems to eliminate the possibility of ever forming associations about complex features. If, for example, the system is not provided with explicit "T-detectors", it should not be able to distinguish "T" shapes except as patterns which stimulate both horizontal and vertical line detectors. Yet the octopus can distinguish "C" from "L" shapes (with difficulty) [You58a].

Further, by this dedication of MUs to CUs, the system is prevented from learning about correlations between CU outputs representing different features: e.g., in a task of attacking black "L"s and white circles but not black squares. (This task is in fact learnable by octopuses [BoY56] and in principle by OVSIM).

- In Maldonado's model, short-term memory is implemented by maintaining recently-addressed MUs at low threshold. This will certainly interfere with their ability to process interim images. Yet the optic lobe is active in the visuomotor feedback loop which oversees the grasping of food in between attack decision and actual ingestion. In OVSIM, short-term memory is implemented as a buffer distinct from the interim processing.
- Maldonado's higher lobes function as amplifiers: adjusting the UE+ and UE-, as in OVSIM, but also simply magnifying the current decision and feeding it back to the optic lobe.
- Delay in Maldonado's model is explained as the time during which EP builds up sufficiently to allow a decision to attack to pass threshold. In OVSIM, delay occurs as probabilities are repeatedly polled.
- Output to the motor centres, in Maldonado's model, is from the higher lobes. In *Octopus* and OVSIM, this output comes from the optic lobes, and can continue even after ablation of the higher lobes.

5.6.3. Predictions made by the OVSIM model

OVSIM (and the ADB approach to delay learning) make several predictions which it would be interesting to investigate in *Octopus*:

1. ADB learns in a backward fashion. Assume the system decides to attack at time t_A, at which point the stimulus is visible, and reinforcement arrives D

cycles later at time t_{A+D}, at which point the stimulus is obscured or even out of vision. Then OVSIM will associate the reinforcement with the image corresponding to t_{A+D}; when this is learned, and the attention to that scene drops, it begins to learn about image t_{A+D-1}, and so on backward. Therefore, at some intermediate point, the system will give a higher response to image t_{A+D} than to the true stimulus t_A. In *Octopus*, this would mean that at some point the animal will strongly attack t_{A+D}-type images (if these can be constructed) before it has learned to attack the stimulus t_A consistently.

2. OVSIM buffers images based on predictability of results. Therefore, if the interval between t_A and t_{A+D} is filled with highly predictable images, the learning of t_A should not be affected. However, if this interval is filled with unknown, and hence unpredictable, images, the system may well never learn about t_A. This would be highly informative, and not too difficult, to investigate in *Octopus*.

3. The suspicion has been raised in *Octopus* that the effects observed after vertical lobe ablation are all caused by domination of the UE+, rather than any deficits in learning capability [You58a]. The most apparent effect of vertical lobe ablation is an indiscriminate increase in attacks, and Young found that if reinforcement was withheld from these animals (thus manually lowering UE+), performance improved as weak negative memories were no longer overpowered by UE+ [You58a]. If this is the whole story, then animals with vertical lobe ablation have no actual memory deficit, only an inability to express restraining memories.

In OVSIM, the removal corresponding to vertical lobe ablation does actually decrease the capability for delay learning. If the buffer is restricted to store only the single most recent pattern, then the only pattern about which associations can be learned is t_{A+D}. The response to pattern t_A will change as a

function of the overlap (in CC feature space) between t_A and t_{A+D}. This was shown to occur in OVSIM (Section 5.5.4). One implication of this would be that the system should be more able to learn about the pattern t_A if it is similar to pattern t_{A+D}.[9] This should be possible to investigate in *Octopus*.

4. The assumption made for OVSIM is that attack, once decided on, is indivisible and unstoppable. This is not the case in the real animal: for example, the octopus may approach an object but not go through with actually touching the stimulus. At the same time, there are a few contradictory data suggesting that the octopus cannot stop a full-fledged attack, once begun. If an octopus begins an attack, and the lights in the experimental room are turned out, the animal completes the attack in the dark [MuG88]. More dramatically, if the animal is trained to attack vertical sinusoidal gratings, but not horizontal ones, and then a horizontal grating is shown which is too fine for the octopus to resolve, the animal eventually attacks [MuG88]. Even when the octopus nears the stimulus enough that it can surely detect its error, the attack is completed. This certainly suggests that attack is unstoppable once begun. If it is possible to reverse this decision, it is quite important to discover where this overriding control originates. No such mechanism is included in OVSIM or in Maldonado's model, with the implication that it does not exist, or that it exists outside both the optic lobe and the superior frontal-vertical lobe circuit.

[9] Appendix E shows two sequences, one which is "harder" in this context since pattern t_A and t_{A+D} differ greatly, and one which is "easier" since the patterns are more similar. It is to be expected that an OVSIM without attention-assignment and with buffer of size 1 could learn more about the stimulus from the "easier" sequence.

CHAPTER 6: SUMMARY AND APPRAISAL

6.1. Summary of the Thesis Contributions

The purpose of this thesis has been to develop a class of neural network machines which perform exploratory learning with delayed reinforcement. The requirements leading to this have been stated as three criteria:

- P1. Exploratory learning, with no discrete training period.
- P2. Reinforcement learning, with only a global evaluation of success, rather than supervised learning with each desired output provided.
- P3. Delay learning, where each reinforcement may not be immediate, and where other unrelated or contradictory reinforcements may intervene.

The first contribution described in this thesis was the definition of the MPLN, an extension of Aleksander's RAM-based PLN which can satisfy both P1 and P2. Like the PLN, its learning algorithm only requires a global reinforcement signal, thereby satisfying P2. By allowing more internal states than the PLN, its learning algorithm can be made incremental, so that behaviour changes more smoothly with learning, and learning can therefore be on-line, as the net is shaped to the desired behaviour.

There are several other models which allow for exploratory reinforcement learning. The advantages of using MPLNs in this type of work are the same as those for using MPLNs in general, and accrue from their RAM-based nature. RAM-based nodes tend to be quick to train, are capable of learning any boolean function of their inputs, and typically exist in topologies which require only low interconnectivity, and are therefore more easily implementable than weighted-sum-and-threshold units.

Having developed a model which satisfies P1 and P2, the second contribution of this thesis was the development of a method for using MPLN networks to satisfy

P3. This took the form of Attention-Driven Buffering (ADB).

ADB stores past input/output combinations in a buffer so that they are still available when reinforcement arrives at some later time. The probability that an element remains in this buffer after some time is related to an attention parameter, dependent in turn on how predictable reinforcement to that element is. Highly predictable elements are less likely to enter the buffer, as there is little or nothing more to learn about them; this leaves room for more uncertain ones to remain in the buffer for long periods. In this way, it is possible to maintain a small buffer but to learn over indefinitely long intervals. Section 4.4 showed that the buffer may be as small as 1; it is the attention-assigning function which ultimately determines the maximum delay bridgeable.

The most celebrated work on delay learning is that of Barto and Sutton. In [BSA83], they develop a system which bridges the reinforcement gap by associating events at time t with expected events at time $t+1$ (and so events immediately preceding reinforcement become reinforcers in their own right). This has considerable support from animal data (c.f. [CoR86]), and is an effect which ADB cannot match. In many situations, this secondary reinforcement is a good thing; in the case of a bad (and critical) action followed by several well-chosen (but ineffective) ones, it is not desirable.

ADB depends critically on being able to assign an attention or confidence to each decision to act. The architectures used in Chapters 4 and 5 allowed this. However, ADB can only be used with architectures that do allow this. It would not be straightforward, for instance, to apply ADB to a net consisting of a single pyramid, which could only output a binary act/not-act decision with no indication of confidence. Some convention would have to be adopted, such as sampling output over time, or allowing non-binary output from the topmost node.

Perhaps the largest difficulty with the use of an ADB system is the choice of the attention-assigning function. This was demonstrated to be the single most important parameter governing the behaviour and capability of the ADB system, and yet its choice throughout this thesis has been essentially arbitrary. It seems obvious that the function should be an inverted paraboloid, as this yields the properties necessary: namely, that attention should be highest for middle values (unpredictable outcomes) and lower for extreme (predictably good or bad) values of system output. However, the maximum height and width of this paraboloid are less easy to justify, other than as ad hoc solutions to the problem domain involved. From an engineering point of view this is not a particular difficulty, as the most important goal is merely to solve the problem. From the point of view of cognitive modelling, it would be preferable to have an attention-assigning function which was more universally powerful and general, and perhaps which the system itself could adjust over time.

The third contribution of this thesis was the application of an MPLN/ADB system to a problem of simulating the operant conditioning behaviour of the visual attack learning system in *Octopus*. The octopus is a well-studied creature with a highly distributed nervous system, capable of learning to attack some objects and reject others. This involves remembering the initial stimulus image while the object is approached, grasped and placed in the mouth – only then might the reinforcement of taste signals arrive.

The ADB system to model this behaviour, OVSIM, showed several similarities with *Octopus* learning, of which the most noteworthy were:

- negatively accelerated learning curve for rejection and discrimination tasks
- fall in delay to attack with positive trials; and an increase in delay with negative trials

- an unspecific effect (UE) of positive trials to increase the likelihood of attack to *any* stimulus, and of negative trials to decrease it

These data provide strong support for a model of *Octopus* as a stochastic machine, where learning may be defined as adjusting the probability of attacking in response to reinforcement obtained.

When the ADB attention-assigning function and UE- in OVSIM were damaged, OVSIM showed behaviour comparable with that of the octopus after vertical lobe ablation. Previously acquired knowledge was still accessible; new learning was more difficult, but close spacing of trials helped, as it was manual provision of a short-term memory in the absence of attention-driven buffering. Trials without reinforcement result in quite good performance, implying learning occurs after damage, but its expression is masked. Again, this is circumstantial evidence that the vertical lobe in *Octopus* may be performing what in OVSIM is easily identifiable as attention assignment and UE- adjustment.

6.2. Appraisal of Physiological Relevance

6.2.1. Objectives

The first question to be asked when addressing the physiological relevance of the machines studied in this thesis is, do they perform a meaningful task? Do animals actually have to perform under conditions of P1, P2 and P3?

P1, which requires exploratory learning, is clearly achieved by every animal which learns. It is possible to imagine an animal which had two phases in its life: one, where it merely sat passively and was taught how to behave, and a second, where it performed accordingly but was no longer capable of learning new behaviours. But this defeats the purpose of having learning ability, which is to adapt to new or changing environments. An animal which learned passively and in a

distinct phase would have no advantage over an animal which simply had all of its behaviours genetically pre-wired. In fact, this seems never to be the case, as all animals tested so far can at least habituate – learning to stop responding to stimuli which repeatedly prove irrelevant [Wel68, p. 158] – and therefore can learn to adapt their behaviour to the requirements of their environments.

P2 declares that reinforcement signals are global measures of success rather than specific target outputs. In most man-made learning machines, a "teacher" exists to instruct each component in its required response to the inputs. This may have parallels in rote school learning and in reflexes, but it is quite improbable that this happens in general sensorimotor learning – for the simple reason that the teacher would have to predict "how each motoneuron involved in the task should respond to each afferent volley, and would have to be able to provide these motoneurons with this information....it is hard to imagine where such detailed information would come from" [Bar87].

P3, learning across delays in which contradictory reinforcements may arrive, is the most demanding criterion. It was shown in Chapter 5 that even invertebrates such as *Octopus* can be conditioned when delay between decision to act and arrival of reinforcement may take seconds. There is also a body of literature documenting experiments with animals such as rat and quail which can learn to avoid food when ingestion was followed with irradiation to produce nausea [Wal87, p. 233], or to prefer food if followed by thiamine injections to remedy a deficiency [GEY67], etc. Even if these "reinforcers" are delayed by several hours [Fel81], the animal learns to seek or avoid the food. It is therefore apparent that these animals have means of accomplishing delay learning as defined by P3.

6.2.2. Methods

The next question to consider is how animals accomplish these tasks, and whether their methods can be said to bear any similarity to ADB.

It is accepted that animals have some form of short-term memory (or "working" memory) as this is how the gap until delayed reinforcement arrives must be bridged. In human beings, STM is often thought of as taking the form of a limited-size buffer, with capacity for some 5-7 items at once – this being the maximum number a person can remember at once, if allowed uninterrupted rehearsal of the items.[1] However, once rehearsal is interrupted, items in STM have a half-life of some 10-15 seconds before they are lost [Joh82]. ADB also involves a limited-capacity buffer, in which items are lost when new, higher-attention items intervene.

In humans, at least, it seems that rehearsal of items makes them more likely to enter long-term memory or to be retrievable from long-term memory [Car86, p. 598]. In ADB, as in many models involving STM [c.f. Gro71], items in STM have correlates in LTM which are strengthened with each cycle that the item remains in the buffer.[2] This sort of approach requires an STM mechanism which maintains activity in relevant cells for a reasonably long period of time. One way in which this might be achieved is by reverberation: a cycle of cells which activate one another to form a circuit which may be active until something interrupts the sequence.

---

[1] There is increasing evidence that the view of STM as holding "7-plus-or-minus-2" items is oversimplified. These items can be individual letters, words, even sentences. It is possible that there is a short-term store for each sort of information. Baddely and Hitch (1974) [Joh82] conclude that STM must contain at least three components: a central executive, a short-term visuo-spatial store, and a short-term speech store. The speech store can be rehearsed, and therefore has a maximum capacity of as much information as can be verbally rehearsed within some 3 seconds. The visual store cannot be rehearsed verbally, and therefore is limited to contain some 2-3 items.

The ADB mechanisms defined in this thesis associate a part of the buffer with every cell. If different cells in the memory are associated with strongly differing information types, then there would effectively be the possibility of buffering several instances of each information type. This possibility has not been explored in this thesis, primarily because the simulations have only involved a single modality – vision.

[2] In particular, with each cycle that item x remains in the ADB buffer, it is strengthened in LTM by an amount equal to H. In the simulations in this thesis, positive reinforcement often meant $H = 1$ for 10 cycles, while negative reinforcement would set $H = -1$ for 10 cycles. If no reinforcement arrived, H would be set to 0, and there would be no net change to x's LTM representation.

In fact, reverberating circuits have been found in the small nervous systems of some invertebrates [Sel88], typically governing central pattern generation, such as might be needed to produce a rhythmic heartbeat.

In mammalian cortex, such reverberation may also exist. Burns, in 1958 [Car86, p. 605-6] studied isolated but internally intact cortical regions, and found that they were normally quiet, but that a train of electric pulses could cause bursts of activity across the region which might continue as long as 30 minutes. A single large shock to the center of the region could then stop all activity. The suggestion was that the region was reverberating, once started by the initial pulses; the larger burst caused all neurons to fire at once, and then as all were simultaneously in the refractory state, the reverberation was silenced. This does not prove, of course, that such reverberation is actually the mechanism whereby short term memory is produced; however, events which interrupt electrical activity in the brain, such as blows to the head, electroconvulsive therapy and epilepsy, also have disruptive effects on short term memory [Ste87, p. 350].

It thus appears that animals would be capable of maintaining a STM buffer like that required for ADB. They would also need to be capable of executing the attention-assigning function which governs which elements occupy the buffer.

The probability of an item entering the ADB buffer in the first place is related to its attention, which is a measure of how "unpredictable" the item is. This concept of learning being based on a need to resolve unpredictable events is extant in psychological theory. Notably, Kelly's Personal Construct Theory suggests that an animal (or human) has as its chief motivation the desire to resolve ambiguities in its input – and to be able to predict results and anticipate its own next input [Kel55]. Animals have shown that they can learn "for learning's sake" even when there is no other ostensible motivation for forming memories. For example, rats turned loose in a maze will explore, and can later return efficiently to where they saw food when

they weren't hungry [Wal87, p. 151-2].

Such a curiosity-driven theory of learning entails that the animal must be able to recognise these ambiguities when they arise. This means that the animal must have the ability to judge unpredictability, like is provided by the attention-assigning component of an ADB system. It is worth emphasising that this requires not merely a novelty detector, but one which measures a stimulus's unpredictability – a stimulus may be familiar but still ambiguous. As will be discussed below, this is a putative role for the amygdala in mammals.

Another implication of the ADB system is that the buffer is separate from the cells or portion of the cells which perform the ongoing processing and which actually maintain LTM (the MPLNs, in the case of the simulations of Chapters 4 and 5). The alternative, as in Maldonado's model of *Octopus* (Section 5.6.2), is for STM to maintain activated cells at a low threshold. This immediately raises the problem that the system must be able to keep processing incoming data in the meantime.[3]

As it happens, the idea of keeping STM in a buffer separate from LTM and from ongoing processing is not incompatible with physiological data. A popular current hypothesis of memory formation involves long-term potentiation (LTP), which is defined as a long-lasting increase in synaptic efficacy produced by high-frequency stimulation of afferents [Ber84]; its counterpart, long-term depression (LTD) could produce a decrease. LTP mechanisms are still a subject for research, but there are several theories which posit a change which does not participate in the short-term operation of the neuronal circuit [e.g., LyB84]. This would allow short- and long-term processing to co-exist in the cell with minimal mutual interference.

A final major implication of the ADB system which must be reconciled with physiological data is its prediction of backward conditioning. In normal classical

[3] It entails a further problem noted by Grossberg [Gro87, v.1, p.182-3]: if only unpredictable events enter STM, how is it that very predictable stimuli can still be processed?

conditioning, an unconditioned stimulus (UCS) is chosen which reliably elicits a response (R). Then if a conditioned stimulus (CS) is repeatedly presented just before the UCS, it will eventually come to elicit R even if the UCS is no longer presented. For maximum effect, the CS should be presented some fixed time before the UCS.[4] If the delay is longer, the CS is irrelevant; if the delay is less, the CS has no value as a predictor of the UCS and will not be learned [Klo88]. If the CS onset actually precedes UCS onset, a set-up termed backward conditioning, the CS may actually come to inhibit the elicitation of R [Klo88], although the effects may be quite complex.

However, backward operant conditioning can actually result in formation of associations. Hudson (1939, 1950) [Wal87] electrified a food cup with a striped pattern; after a week the rats no longer ate from the cup but piled sawdust over it as if to hide the pattern and retreated. If the experimental setup was changed so that as the shock was administered, the lights went out and the cup was removed, many of the rats failed to learn the association. This implies that the association was formed in the period immediately following the UCS administration [Wal87, p. 88-9]. Hudson also found that if pipe cleaners were dropped into the cage just after the shock, the rats would selectively avoid those [Wal87, p. 89]. Similarly, Keith-Lucas and Guttman introduced a toy animal into the cage within ten seconds after the shock, and found that the rats would develop a conditioned aversion to the toy [Wal87, p. 89]. All of these experiments indicate that associations can be formed in the few seconds immediately following the UCS.

The ADB system will associate reinforcement with whatever is currently in its buffer. If the items resident have high attention, they will be reinforced and items occurring just after the UCS but unable to enter the buffer will be ignored. If those

[4] The actual optimal timing is dependent both on the species being trained and the response being tested.

new items can get into the buffer, the reinforcement will be attributed to them, and backward conditioning will occur. This is possibly analogous to the animal case.

6.2.3. Mechanisms

Finally, it is worth a short consideration of where the ADB operations might be occurring in the mammalian brain. The hippocampus, amygdala, hypothalamus, orbitofrontal cortex and inferior temporal cortex are all highly interconnected in mammals, and all seem to be implicated in learning responses to reinforcement, but not in producing those responses once learned.

The hippocampus in particular has been highly studied and is known to be involved in a considerable variety of tasks. These include:

- Ability to learn *not* to perform a trained response [Raw85]
- Maintenance of short-term or "working" memory [Joh82]
- Trace conditioning – with hippocampal lesions, rats can still perform simple classical conditioning [Car86, p. 591], but are unable to learn when there is a delay between lever press and food or between tone and food [Raw85]

These functions, mostly noticeable by their absence after hippocampal damage, are quite parallel to those seen in *Octopus* after lesions to the vertical lobe. If the vertical lobe in *Octopus* can be thought of as performing some of the operations necessary for ADB, perhaps the hippocampus is also involved in these operations.

There are a variety of sophisticated theories of hippocampal function. It may be an intermediate-term buffer for short term memory, especially where stimulus representations must be maintained until results arrive [Raw85]; this of course is one operation required by ADB. It may, instead of being a buffer itself, facilitate rehearsal of representations elsewhere in associative memory [Raw85]; again, this is not incompatible with ADB (see Section 4.3). The hippocampus may work as a

comparator of experienced events with predictions [McN82]. This task is one which is not explicitly carried out by the ADB machine as formalised in this thesis, but is very much the sort of mechanism postulated by Sutton and Barto for their Adaptive Critic Element [BSA83]. Rolls has even found specific hippocampal cells which fire more strongly on the first presentation of an object than on subsequent presentations [Rol89], clearly performing some measure of novelty detection.

A second brain region which may have some bearing on the ADB approach is the amygdala. This region gets highly processed input from the sensory systems and the cortex, and outputs to motor, autonomic, associative and limbic cortex [Rol90]. With amygdaloid lesions, among other effects, the subject becomes unable to learn avoidance and will examine objects excessively, as if unable to decide what to do with them [Rol90]. Experimentally, there are neurons in the amygdala which never respond to negative stimuli and do respond to positive or novel stimuli – thus maintaining images of the latter as active while filtering out the former [WiR90]. Wilson and Rolls suggest that "amygdala neurons would signal that representations should be retained if a visual stimulus has been previously associated with a positive reinforcer ... or attention should be paid to that stimulus" [WiR90]. This is in many ways analogous to the problem of attention-assignment within an ADB system which depends upon determining the predictability of an input, and novelty is certainly related in most circumstances to predictability.

6.3. Future Work

There are three obvious directions in which the work presented in this thesis could be furthered: extending and completing the OVSIM model, directing attention to parallels with mammalian brains, and simply improving the utility of ADB as a computing tool.

In the last case, the system can be endowed with several capabilities, ignored in this thesis, but which allow for more complicated behaviour. Most notably, the attention function presented here has been a simple, automatically assigned, and static function. This is certainly not the case in human attention, and need not be the case in a computer. Humans seem to be capable of artificially boosting (or lowering) attention as a result of higher level commands – we can make a conscious effort to "pay attention" to something, and we can make a similar conscious decision to "memorise" something. In terms of ADB, this would be a higher-level executive controller which could step in and assign high attention to some inputs – overriding the usual assignment routines. The phenomenon of rehearsal in humans is also much more complex than has been considered here. For example, there seems to be a verbal rehearsal route (the limit of STM seems to be not so much "7 items" as the amount of material which a subject can repeat to himself within a cycle time of about 3 seconds). This route would therefore have to include not merely the working memory, but also auditory speech areas, for example. All of these issues could be explored with further research on ADB.

The ADB model could also be examined much more completely than it has been here as a hypothesis of mammalian learning. In order to make such a comparison feasible, the system would have to be able to mimic data from studies on hippocampus and other regions.

Finally, the OVSIM presented in Chapter 5 is by no means a complete account of the *Octopus*. To begin with, the visual attack learning system is capable of much more sophisticated discrimination, generalisation, and output activity. The octopus also has a highly developed touch learning system, which could be modelled and made to interact with the visual system in the way it certainly does in the normal animal. The touch learning system is also capable of transfer between arms and between optic lobes. These phenomena are not completely understood, and if the

OVSIM could be made to accomplish these tasks, it would provide at least a working hypothesis for the experimenter to investigate.

Probably the most interesting aspect of this work has been that a system which was originally developed from purely pragmatic engineering specifications should turn out to have some similarities to the solution found by nature in the *Octopus*. If it were to turn out that these similarities held in detail in higher brains as well, there would be a rather dramatic conclusion to be drawn: that there may actually only be one fairly constrained way in which to accomplish the task of learning. On the other hand, if the parallels between ADB and the *Octopus* are eventually proven to be spurious, then the opposite conclusion holds, and there may in fact be many different ways in which to construct an intelligent machine.

This is an intriguing question. But it seems clear that the approach to answering it must continue on two fronts: the bottom-up approach, whereby physiologists and psychologists continue to provide data on how brains accomplish their tasks, and the top-down approach, in which engineers and cognitive scientists propose models to satisfy these data. From this point of view, it is just as important to find out which models are *not* satisfactory and why, as this will still provide information. Whether ADB has any validity or not as a psychological model, therefore, I hope that continued work with it will prove to contribute at least some small clue as to how intelligent learning systems can one day be constructed.

REFERENCES

[ABJ89] 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.

[AbP87] Abu-Mostafa, Y.; Psaltis, D. (1987) Optical neural computers. *Scientific American*, **256**(3): 66-73.

[AbS85] Abu-Mostafa, Y.; St. Jacques, J. (1985) Information capacity of the Hopfield model. *IEEE Trans. on Information Theory*, **IT-31**: 461-464.

[AcL90] Ackley, D.; Littman, M. (1990) Learning from natural selection in an artificial environment. *Proc. International Joint Conference on Neural Networks*, Washington, D. C., vol. I, pp. 189-193.

[AlD85] Aleksander, I.; Dobree-Wilson, M. (1985) Adaptive windows for image processing. *IEE Proceedings*, **132E**(5): 233-245.

[Ale88] Aleksander, I. (1988) Logical connectionist systems. In, *Neural Computers* (eds. R. Eckmiller, C. von der Malsburg). Springer-Verlag, Berlin, pp. 189-197.

[Ale89] Aleksander, I. (1989) Canonical neural nets based on logic nodes. *Proc 1st IEE International Conference on Artificial Neural Networks*, London, pp. 110-114.

[Ale90] 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.

[AlS79] Aleksander, I.; Stonham, T. (1979) Guide to pattern recognition using random-access memories. *IEE J. Computers and Digital Tech.*, **2**(1): 29-40.

[AlS90] Al-Alawi, R.; Stonham, T. (1990) Evaluation of the functional capacities of multi-layered logical neural networks. *Proc. INNC-90-PARIS*, Paris, pp. 983.

[ATB84] Aleksander, I.; Thomas, W.; Bowden, P. (1984) WISARD – A radical step forward in image recognition. *Sensor Review*, 4(3): 120-124.

[Aus88] Austin, J. (1988) Grey-scale n-tuple processing. *BPRA 4th International Conference on Pattern Recognition*, Cambridge.

[BaA85] Barto, A.; Anandan, P. (1985) Pattern-recognising stochastic learning automata. *IEEE Trans. on Systems, Man and Cybernetics*, **SMC-15**(13): 360-375.

[Bar85] Barto, A. (1985) Adaptive neural networks for learning control: Some computational experiments. *Proc. IEEE Workshop on Intelligent Control*, Rensselaer, pp. 170-175.

[Bar86] Barto, A. (1986) Game-theoretic cooperativity in networks of self-interested units. In, *Neural Networks for Computing* (ed. J. Denker). American Institute of Physics, New York, pp. 41-46.

[Bar87] Barto, A. (1987) An approach to learning control surfaces by connectionist systems. In, *Vision, Brain and Cooperative Competition* (eds. M. Arbib, A. Hanson). MIT Press, London, pp. 665-701.

[BaS81] Barto, A.; Sutton, R. (1981) Landmark learning: An illustration of associative search. *Biological Cybernetics*, **42**: 1-8.

[BaS82] Barto, A.; Sutton, R. (1982) Simulation of anticipatory responses in classical conditioning by a neuron-like adaptive element. *Behavioural Brain Research*, 4: 221-235.

[Ber84] Berger, T. (1984) Long-term potentiation of hippocampal synaptic transmission affects rate of behavioral learning. *Science*, **224**: 627-630.

[BFF89] Bisset, D.; Filho, E.; Fairhurst, M. (1990) A comparative study of neural network structures for practical application in a pattern recognition

environment. *Proc. 1st IEE International Conference on Artificial Neural Networks*, London, pp. 378-382.

[Bla75] Blakemore, C. (1975) Central visual processing. In, *Handbook of Physiology* (eds. M. Gazzaniga, C. Blakemore). Academic Press, London, pp. 241-268.

[Bla88] Blakemore, C. (1988) *The Mind Machine*. BBC Books, London.

[BlB59] Bledsoe, W.; Browning, I. (1959) Pattern recognition and reading by machine. *Proc. Eastern Joint Computer Conference*, Boston, pp. 225-232.

[Blo62] Block, H. (1962) The perceptron: A model for brain functioning I. *Reviews of Modern Physics*, **34**: 123-135.

[Boo88] Booker, L. (1988) Classifier systems that learn internal world models. *Machine Learning*, **3**: 161-192.

[BoY55] Boycott, B.; Young, J. (1955) A memory system in Octopus vulgaris Lamarck. *Proc. Royal Society of London*, **B143**: 449-480.

[BoY56] Boycott, B.; Young, J. (1956) Effects of interference with the vertical lobe on visual discriminations in Octopus vulgaris Lamarck. *Proc. Royal Society of London*, **B146**: 439-459.

[Boy67] Boycott, B. (1967) Learning in the octopus. In, *Psychobiology: The Biological Basis of Behaviour* (eds. J. McGaugh, N. Weinberger, R. Whalen). W. H. Freeman, San Francisco, pp. 132-140.

[BSA83] Barto, A.; Sutton, R.; Anderson, C. (1983) Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man and Cybernetics*, **SMC-13**(5): 834-851.

[BSB81] Barto, A.; Sutton, R.; Brouwer, P. (1981) Associative search network: A reinforcement learning associative memory. *Biological Cybernetics*, **40**: 201-211.

[CaG87] Carpeter, G.; Grossberg, S. (1987) ART2: Self-organisation of stable category recognition codes for analog input patterns. *Applied Optics*, **26**(23): 4919-4930.

[CaG88] Carpenter, G.; Grossberg, S. (1988) The ART of adaptive pattern recognition by a self-organising neural network. *IEEE Computer*, **21**(3): 77-88.

[CaP87] Carnevalli, P.; Patarnello, S. (1987) Exhaustive thermodynamical analysis of Boolean learning networks. *Europhysics Letters*, 4(10): 1199-1204.

[Car86] Carlson, N. *The Physiology of Behavior*. Allyn and Bacon, London.

[CeP89] Cecconi, F.; Parisi, D. (1989) Networks that learn to predict where the food is and also to eat it. *Proc. International Joint Conference on Neural Networks*, Washington, D. C., vol. II, p. 624 (abstract only).

[CGT90] Clarkson, T.; Gorse, D.; Taylor, J. (1990) Hardware realisable models of neural processing. *Proc. 1st IEE International Conference on Artificial Neural Networks*, London, pp. 242-246.

[Che90] Chen, V. (1990) Problem-solving by using reinforcement learning neural nets. *Proc. International Joint Conference on Neural Networks IJCNN-90-WASH*, Washington, D. C., vol. II, pp. 583-586.

[ChS89] Churchland, P.; Sejnowski, T. (1989) Neural representation and neural computation. In, *Neural Connections and Mental Computation* (eds. L. Nadel, L. Cooper, P. Culicover, R. Harnish). MIT Press, London, pp. 15-48.

[ChV89] Cherassky, V.; Vassilas, N. (1989) Performance of backpropagation networks for associative database recall. *Proc. International Joint Conference on Neural Networks IJCNN-90-WASH*, Washington, D. C., vol. I, pp. 77-84.

[CoR86] Colwill, R.; Rescorla, R. (1986) Associative structures in instrumental learning. In, *The Psychology of Learning and Motivation: Advances in Research and Theory* (ed. G. Bower). Academic Press, London, vol. 20.

[CrA86] Crick, F.; Asanuma, C. (1986) Certain aspects of the anatomy and physiology of the cerbral cortex. In, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (eds. J. McClelland, D. Rumelhart). MIT Press, London, vol. 2, pp. 372-289.

[deC86] de Callatay, A. (1986) *Natural and Artificial Intelligence.* Elsevier Science, New York, pp. 260-271.

[DGZ87] Derrida, B.; Gardner, E.; Zippelius, A. (1987) An exactly solvable asymmetric neural network model. *Europhysics Letters*, 4(2), pp. 167-173.

[Dor68] Doran, J. E. (1968) Experiments with a pleasure-seeking automaton. *Machine Intelligence* (ed. D. Mitchie). Edinburgh University Press, Edinburgh, vol. 3, pp. 195-216.

[Fel81] Feldman, J. (1981) A connectionist model of visual memory. In, *Parallel Models of Associative Memory* (eds. G. Hinton, J. Anderson). Lawrence Erlbaum, Hillsdale, NJ, pp. 49-81.

[Ful89] Fulcher, E. (1989) Neural networks: A testbed for new conjectures. BSc Thesis, Middlesex Polytechnic.

[GEY67] Garcia, J.; Ervin, F.; Yorke, C.; Koelling, R. (1967) Conditioning with vitamin injections. *Science*, **185**: 824-831.

[GoS88] Gorman, R.; Sejnowski, T. (1988) Learned classification of sonar targets using a massively parallel network. *IEEE Trans. on Acoustics, Speech and Signal Processing*, **36**(7): 1135-1140.

[GoT88] Gorse, D.; Taylor, J. (1988) On the equivalence and properties of noisy neural and probabilistic RAM nets. *Physics Letters A*, **131**(6): 326-332.

[GoT90] Gorse, D.; Taylor, J. (1990) Training strategies for probabilistic rams. *Parallel Processing in Neural Systems and Computers* (eds. R. Eckmiller, G. Hartmann, G. Hauske). Elsevier Science, Amsterdam, pp. 161-164.

[Gro71] Grossberg, S. (1971) On the dynamics of operant conditioning. *Journal of Theoretical Biology*, **33**: 225-255.

[Gro80] Grossberg, S. (1980) How does a brain build cognitive code? *Psychological Review*, **87**: 1-51.

[Gro87] Grossberg, S. (1987) *The Adaptive Brain.* Elsevier Science, New York.

[GuM90] Guha, A.; Mahur, A. (1990) Setpoint control based on reinforcement learning. *Proc. International Joint Conference on Neural Networks IJCNN-90-WASH*, Washington, D. C., vol. II, pp. 511-514.

[HAR84] Hartwick, E.; Ambrose, R.; Robinson, S. (1984) Den utilization and the movements of tagged Octopus dofleini. *Marine Behavior and Physiology*, **11**(2): 95-110.

[HaL84] Hartline, P.; Lange, G. (1984) Visual systems of cephalopods. In, *Comparative Physiology of Sensory Systems* (eds. L. Bolis, R. Keynes, S. Maddrell). Cambridge University Press, London, pp. 335-355.

[Heb49] Hebb, D. (1949) *The Organization of Behavior.* Chapman and Hall, London.

[Hop82] Hopfield, J. (1982) Neural networks and physical systems with emergent collective computational abilities. *Proc. National Academy of Science USA*, **79**(8): 2554-2558.

[Hul32] Hull, C. (1932) The goal-gradient hypothesis and maze learning. *Psychological Review*, **39**: 25-43.

[Hun88] Hunter, L. (1988) Some memory but no mind. *Behavioral and Brain Sciences*, **11**(1): 37.

[Joh82] Johnson-Laird, P. (1988) *The Computer and the Mind.* Fontana Paperbacks, London.

[Jor86] Jordan, M. (1986) Attractor dynamics and parallelism in a connectionist sequential machine. *Proc. 8th Annual Conference of Cognitive Science Society*, pp. 531-546.

[KaA87] Kan, W.; Aleksander, I. (1987) A probabilistic logic neuron network for associative learning. *Proc. IEEE 1st Annual International Conference on Neural Networks*, San Diego, pp. 541-548.

[Kan76] Kandel, E. (1976) *Cellular Basis of Behavior: An Introduction to Behavioral Neurobiology*. W. H. Freeman, San Francisco.

[Kan88] Kan, W. (1988) A probabilistic logic neural network for associative learning. PhD Thesis, Imperial College, University of London.

[Kel55] Kelly, G. (1955) *The Theory of Personal Constructs*. Norton, New York.

[Klo86] Klopf, A. (1986) A drive-reinforcement model of single neuron function: An alternative to the Hebbian model. In, *Neural Networks for Computing* (ed. J. Denker). American Institute of Physics, New York, pp. 265-270.

[Klo88] Klopf, A. (1988) A neuronal model of classical conditioning. *Psychobiology*, **16**(2): 85-125.

[Koh88] Kohonen, T. (1988) The "neural" phonetic typewriter. *Computer*, **21**(3): 11-22.

[Koh90] Kohring, G. (1990) Finite-state neural networks: A step towards the simulation of very large systems. (Preprint obtained from author).

[KuR89] Kuperstein, M.; Rubenstein, J. (1989) Implementation of an adaptive neural controller for sensory-motor coordination. *Proc. International Joint Conference on Neural Networks IJCNN-90-WASH*, Washington, D. C., vol. II, pp. 305-310.

[Lan85] Langley, P. (1985) Learning to search: From weak methods to domain-specific heuristics. *Cognitive Science*, **9**: 217-260.

[LeC86] LeCun, Y. (1986) Learning processes in an asymmetric threshold network. In, *Disordered Systems and Biological Organization* (eds. E. Bienenstock, F. Fogelman Souli, G. Weisbuch). Springer-Verlag, Berlin.

[LeM89] Leaver, R.; Mars, P. (1989) Stochastic computing and reinforcement neural networks. *Proc. 1st IEE International COnference on Artificial Neural Networks*, London, pp. 163-170.

[LGH89] LeCun, Y.; Galland, C.; Hinton, G. (1989) GEMINI: Gradient estimation through matrix inversion after noise injection. *Advances in Neural Information Processing Systems I* (ed. D. Touretzky). Morgan Kaufman, San Mateo, CA, pp. 141-148.

[LyB84] Lynch, G.; Baudry, M. (1984) The biochemistry of memory: A new specific hypothesis. *Science*, **224**: 1057-1063.

[Mal63] Maldonado, H. (1963) The visual attack learning system in Octopus vulgaris. *Journal of Theoretical Biology*, **5**: 470-488.

[MaL85] Mather, J.; Lethbridge, U. (1985) Behavioral interactions and activity of captive Eledone moschata: Laboratory investigation of a "social" octopus. *Animal Behvaior*, **33**(4): 1138-1144.

[Mar82] Marr, D. (1982) *Vision*. W. H. Freeman, San Francisco.

[Mar88] Martland, D. (1988) Adaptation of Boolean networks using back-error propagation. (Preprint obtained from author).

[McA88] McAulay, A. D. (1988) Adaptive 2-D tracking with neural networks. *Abstracts of 1st Annual INNS Meeting*, Boston, p. 457 (abstract only).

[McE86] 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.

[McN82] McNaughton, N. (1982) Is the hippocampus a store, intermediate or otherwise? *Behavioral Brain Sciences*, **8**: 508-509.

[McP43] McCulloch, W.; Pitts, W. (1943) A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, **5**: 115-133.

[Mel88] Mel, B. (1988) MURPHY: A robot that learns by doing. In, *Neural Information Processing Systems* (ed. D. Anderson). American Institute of Physics, New York, pp. 544-553.

[MiC68] Mitchie, D.; Chambers, R. (1968) BOXES: An experiment in adaptive control. In, *Machine Intelligence 2* (eds. E. Dale, D. Mitchie). Oliver and Boyd, Edinburgh, pp. 137-152.

[MiP69] Minsky, M.; Papert, S. (1969) *Perceptrons: An Introduction to Computational Geometry*. MIT Press, London (2nd edition, 1989).

[MKS88] Miyamoto, H.; Kawato, M.; Setoyama, T.; Suzuki, R. (1988) Feedback-error-learning neural network for trajectory control of a robotic manipulator. *Neural Networks*, **1**(3): 251-265.

[MuG88] Muntz, W.; Gwyther, J. (1988) Visual acuity in Ocotpus pallidus and Octopus australis. *Journal of Experimental Biology*, **134**: 119-129.

[MyA88] Myers, C.; Aleksander, I. (1988) Learning algorithms for probabilistic logic nodes. *Abstracts of 1st Annual INNS Meeting*, Boston, p. 205 (abstract only).

[Mye87] Myers, C. (1987) Training strategies for the PLN. Imperial College Neural Computing Group Internal Report NCG/87/02.

[Mye88] Myers, C. (1988) The number of functions computed by PLN trees. Imperial College Neural Systems Engineering Internal Report NSE/88/02,3.

[Mye89a] Myers, C. (1989) Output functions for probabilistic logic nodes. *Proc. 1st IEE International Conference on Artificial Neural Networks*, London, pp. 310-

314.

[Mye89b] Myers, C. (1989) Temporal credit assignment: Adaptive learning when results are delayed and interleaved in time. Imperial College Neural Systems Engineering Internal Report NSE/89/01.

[Mye89c] Myers, C. (1989) Temporal credit assignment II: Non-independent memories. Imperial College Neural Systems Engineering Internal Report NSE/89/02.

[Mye90] Myers, C. (1990) Reinforcement learning when results are delayed and interleaved in time. *Proc. INNC-90-PARIS*, Paris, pp. 860-863.

[NaN67] Nagumo, J.; Noda, A. (1967) A learning method for system identification. *IEEE Trans. on Automatic Control*, AC-12: 282-287.

[New88] Newman, C. (1988) Memory capacity in neural networks: Rigorous lower bounds. *Neural networks*, 1(3): 223-238.

[NgW89] Nguyen, D.; Widrow, B. (1989) The truck backer-upper: An example of self-learning in neural networks. *Proc. International Joint Conference on Neural Networks IJCNN-90-WASH*, Washingtion, D. C., vol. II, pp. 357-363.

[PaC88] Patarnello, S.; Carnevalli, P. (1990) Learning to predict the consequences of one's own actions. *Parallel Processing in Neural Systems and Computers* (eds. R. Eckmiller, G. Hartmann, G. Hauske). Elsevier Science, Amsterdam, pp. 237-240.

[Par85] Parker, D. (1985) Learning logic. Technical Report TR-87, Centre for Computational Research in Economics and Management Science, MIT, Cambridge, MA.

[PoC90] Procino, D.; Collins, J. (1990) An Application of neural networks to the guidance of free-swimming submersibles. *Proc. International Conference on Neural Networks IJCNN-90-WASH*, vol. II, pp. 417-420.

[PPH88] Psaltis, D.; Park, C.; Hong, J. (1988) Higher-order associative memories and their optical implementations. *Neural Networks*, 1(2): 149-163.

[PSY87] Psaltis, D.; Sideris, A.; Yamamuro, A. (1987) Neural controllers. *Proc. IEEE 1st Annual Conference on Neural Networks*, San Diego, pp. 551-558.

[Raw85] Rawlins, J. (1985) Associations across time: The hippocampus as a temporary memory store. *Behavioral and Brain Sciences*, 8: 479-496.

[RHW86a] 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.

[RHW86b] Rumelhart, D.; Hinton, G.; Williams, R. (1986) Learning representations by back-propagating errors. *Nature*, 323: 533-536.

[Rol89] Rolls, E. (1989) Function of neuronal networks in the hippocampus and neocortex in memory. In, *Neural Models of Plasticity: Experimental and Theoretical Approaches* (eds. J. Byrne, W. Berry). Academic Press, San Diego, pp, 240-265.

[Rol90] Rolls, E. (1990) A theory of emotion and its application to understanding the neural basis of emotion. Preprint obtained from author, to appear in *Cognition and Emotion.*

[Ros58] Rosenblatt, F. (1958) The perceptron: A probabilistic model for information storage and organization in the brain. *Psychology Review*, 65: 386-408.

[Ros62] Rosenblatt, F. (1962) *Principles of Neurodynamics.* Spartan Books, New York.

[Sam63] Samuels, A. (1963) Some studies in machine learning using the game of checkers. In, *Computers and Thought* (eds. E. Feigenbaum, J. Feldman). McGraw-Hill, New York, pp. 71-105.

[SaS89] Saerens, M.; Soquet, A. (1989) A neural controller. *Proc. 1st IEE International Conference on Artificial Neural Networks*, London, pp. 211-215.

[Sch90a] Schmidhuber, J. (1990) Recurrent networks adjusted by adaptive critics. *Proc. International Joint Conference on Neural Networks IJCNN-90-WASH*, Washington, D. C., vol. I, pp. 719-722.

[Sch90b] Schmidhuber, J. (1990) Networks adjusting networks. *Proc. Distributed Adaptive Neural Information Processing* (eds. J. Kinderman, A. Linden), St. Augustin, 1989.

[Sej86] Sejnowski, T. (1986) Open questions about computation in the cerebral cortex. In, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (eds. J. McClelland, D. Rumelhart). MIT Press, London, vol. 2, pp. 372-389.

[Sel88] Selverston, A. (1988) A consideration of invertebrate central pattern generators as computation databases. *Neural Networks*, 1(2): 109-117.

[SeR86] Sejnowski, T.; Rosenberg, C. (1986) NETtalk: A parallel network that learns to read aloud. JHU Technical Report JHU/EECS-86/01.

[Sha89a] Shapiro, J. (1989) Hard learning in boolean neural networks. In, *New Developments in Neural Computing* (eds. J. Taylor, C. Mannion). Adam Hilger, Bristol, pp. 125-132.

[Sha89b] Sharkey, N. (1989) A PDP learning approach to natural language understanding. In, *Neural Computing Architectures* (ed. I. Aleksander). North Oxford Academic, London, pp. 92-116.

[ShM88] Shepanski, J.; Macy, S. (1988) Teaching artificial neural systems to drive: Manual training techniques for autonomous systems. In, *Neural Information Processing Systems* (ed. D. Anderson). American Institute of Physics, New York, pp. 693-700.

[Ste87] Stein, J. (1987) *An Introduction to Neurophysiology*. Blackwell Scientific, London.

[Sto89] Stork, D. (1989) Is backpropagation biologically plausible? *Proc. International Joint Conference on Neural Networks*, Washington D. C., vol. II, pp. 241-246.

[SuB81] Sutton, R.; Barto, A. (1981) Toward a modern theory of adaptive networks: Expectation and prediction. *Psychology Review*, **88**(2): 135-170.

[Sut57] Sutherland, N. (1957) Visual discrimination of orientation and shape by the octopus. *Nature*, **179**: 11-13.

[Sut63] Sutherland, N. (1963) Shape discrimination and receptive fields. *Nature*, **197**: 118-122.

[Sut88] Sutton, R. (1988) Learning to predict by the methods of temporal differences. *Machine Learning*, **3**: 9-44.

[TeS88] Tesauro, G.; Sejnowski, T. (1988) A "neural" network that learns to play backgammon. In, *Neural Information Processing Systems* (ed. D. Anderson). American Institute of Physics, New York, pp. 794-803.

[TFL89] 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.

[ToW88] Tolat, V.; Widrow, B. (1988) An adaptive neural net controller with visual inputs. *Abstracts of 1st Annual INNS Meeting*, Boston, p. 362 (abstract only).

[Vid88] Vidal, J. (1988) Implementing neural nets with programmable logic. *IEEE Trans. on Acoustics, Speech and Signal Processing*, **36**(7): 1180-1190.

[WaG89] Wang, J.; Grodin, R. (1989) Novel training algorithm for limited connection networks. *Proc. 1st IEE International Conference on Artificial Neural Networks*, London, pp. 387-389.

[Wal87] Walker, S. (1987) *Animal Learning: An Introduction.* Routledge and Kegan Paul, London.

[Wel59a] Wells, M. (1959) Functional evidence for neurone fields representing the individual arms within the central nervous system of Octopus. *Journal of Experimental Biology*, **36**: 501-511.

[Wel59b] Wells, M. (1959) A touch-learning centre in Octopus. *Journal of Experimental Biology*, **36**: 590-612.

[Wel68] Wells, M. (1968) *Lower Animals*. Weidenfeld and Nicholson, London.

[Wer74] Werbos, P. (1974) Beyond regression: New tools for predictions and analysis in the behavioral sciences. PhD Thesis, Harvard University, Cambridge, MA.

[Wer90] Werbos, P. (1990) Consistency of HDP applied to a simple reinforcement learning problem. *Neural Networks*, **3**(2): 179-189.

[WeY65] Wells, M.; Young, J. (1965) Split-brain preparations and touch learning in the Octopus. *Journal of Experimental Biology*, **43**: 565-579.

[WGM73] Widrow, B.; Gupta, N.; Maitra, S. (1973) Punish/reward: Learning with a critic in adaptive systems. *IEEE Trans. on Systems, Man and Cybernetics*, **SMC-3**(5): 455-465.

[WiH60] Widrow, B.; Hoff, M. (1960) Adaptive switching circuits. *1960 WESCON Convention Recordings*, part 4.

[WiL88] Wieland, A.; Leighton, R. (1988) Shaping schedules as a method for accelerated learning. *Abstracts of 1st Annual INNS Meeting*, Boston, p. 231 (abstract only).

[Win88] Windecker, R. (1988) Learning of networks of nondeterministic adaptive elements. In, *Neural Information Processing Systems* (ed. D. Anderson). American Institute of Physics, New York, pp. 840-849.

[WiP89] Williams, R.; Peng, J. (1989) Reinforcement learning algorithms as function optimizers. *Proc. International Joint Conference on Neural Networks*, Washington, D. C., vol. II, pp. 89-95.

[WiR90] Wilson, F.; Rolls, E. (1990) The primate amygdala and reinforcement: A dissociation between rule-based and associatively-mediated memory revealed in amygdala neuronal activity.)Preprint obtained from authors).

[WiS64] Widrow, B.; Smith, F. (1964) Pattern-recognizing control systems. In, *Computer and Information Sciences* (eds. J. Tou, R. Wilcox). Spartan Books, New York, pp. 288-317.

[Wit77] Witten, I. (1977) An adaptive optimal controller for discrete-time Markov environments. *Information and Control*, **34**: 286-295.

[WiZ89] Williams, R.; Zipser, D. (1989) A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, **1**(2): 270-280.

[WoS89] Wong, K.; Sherrington, D. (1989) The maximum storage capacity in Boolean associative memories. In, *New Developments in Neural Computing* (eds. J. Taylor, C. Mannion). Adam Hilger, Bristol, pp. 133-140.

[WWB88] Widrow, B.; Winter, R.; Baxter, R. (1988) Layered neural nets for pattern recognition. *IEEE Trans. on Acoustics, Speech and Signal Processing*, **36**(7): 1109-1118.

[Yeu88] Yeung, D. (1988) Supervised learning of action probabilities in associative reinforcement learning. *Proc. 1988 Connectionist Models Summer School* (eds. D. Touretzky, G. Hinton, T. Sejnowski). Morgan Kaufman, San Mateo, CA, pp. 162-171.

[You58a] Young, J. (1958) Effect of removal of various amounts of vertical lobe on visual discrimination in Octopus. *Proc. Royal Society of London*, **B-149**: 441-462.

[You58b] Young, J. (1958) Responses of untrained octopuses to various figures and the effect of removal of the vertical lobe. *Proc. Royal Society of London*, **B-149**: 463-483.

[You60] Young, J. (1960) Unit processes in the formation of representations in the memory of Octopus. *Proc. Royal Society of London*, **B-153**:1-17.

[You64] Young, J. (1964) *A Model of the Brain*, Clarendon Press, Oxford.

[You65a] Young, J. (1965) The nervous pathways for poisoning, eating and learning in Octopus. *Journal of Experimental Biology*, **43**: 581-593.

[You65b] Young, J. (1965) Influence of previous preferences on the memory of Octopus vulgaris after removal of the vertical lobe. *Journal of Experimental Biology*, **43**: 595-603.

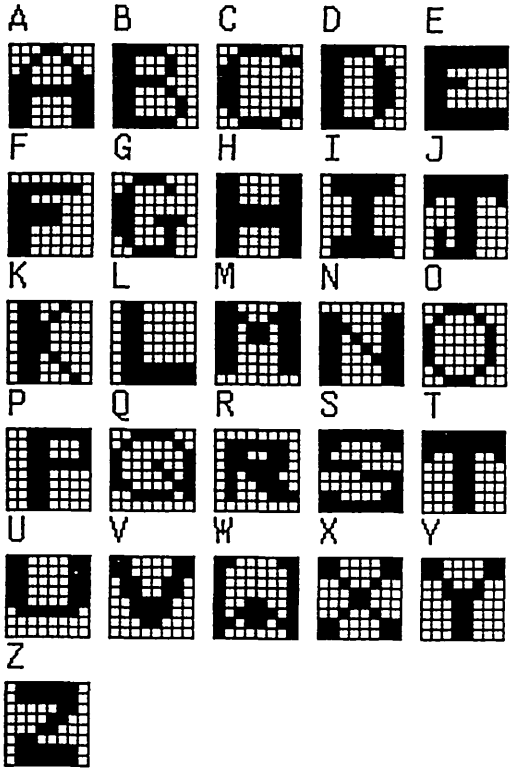
[You68] Young, J. (1968) Reversal of a visual preference in Octopus after removal of the vertical lobe. *Journal of Experimental Biology*, **49**: 413-19.

[You70] Young, J. (1970) Short and long memories in Octopus and the influence of the vertical lobe system. *Journal of Experimental Biology*, **52**: 385-393.

[You71] Young, J. (1971) *The Anatomy of the Nervous System of Octopus vulgaris*. Clarendon Press, Oxford.

APPENDIX A

Below left, the stimulus patterns used for the ADB simulations of Section 4.7; below right, the state transition table for the ADB simulations of Section 4.7 – showing the next state entered if the system outputs a decision to move left, ahead or right from the current state.

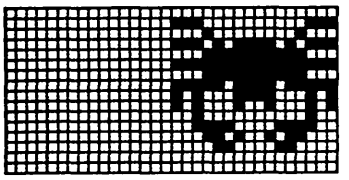


Current State	Next State		
	Left	Ahead	Right
A	B	C	D
B	E	F	G
C	H	I	J
D	K	L	M
E	N	O	P
F	Q	S	T
G	R	U	V
H	W	X	Y
I	Z	A	B
J	D	E	F
K	C	G	H
L	I	J	K
M	L	N	Y
N	O	P	Q
O	R	S	T
P	U	V	W
Q	X	Y	Z
R	A	B	C
S	D	E	G
T	F	H	I
U	J	K	L
V	M	N	O
W	Q	R	Z
X	S	T	U
Y	V	W	X
Z	P	A	M

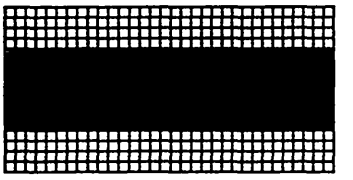
APPENDIX B

The set of stimulus patterns used with the OVSIM simulations of Chapter 5; each is referred to in the text by reference to the 3-character code given here.

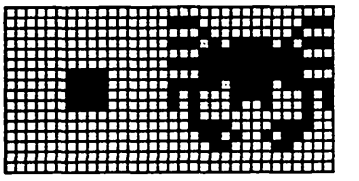
p_C



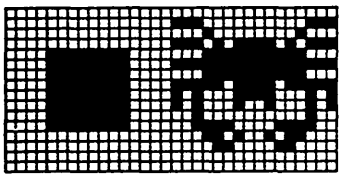
pHH



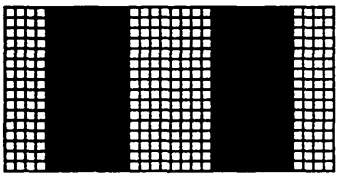
p3C



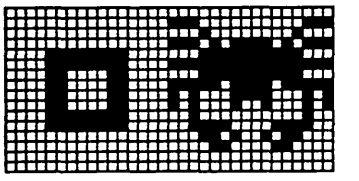
p2C



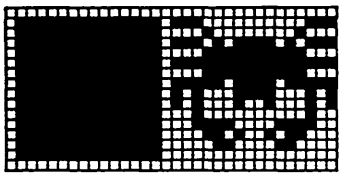
pVV



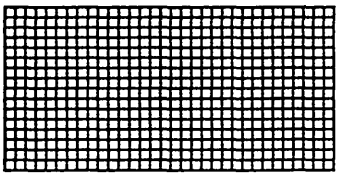
pFC



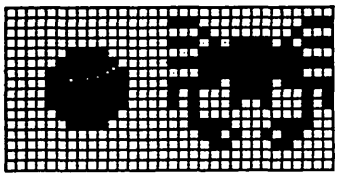
p1C



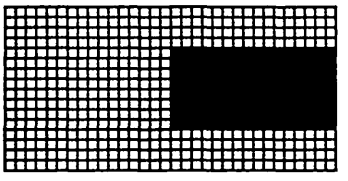
p__



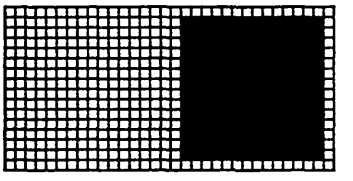
pcC



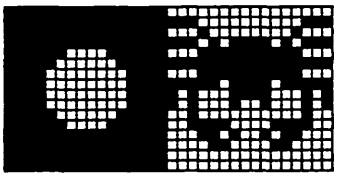
p_H



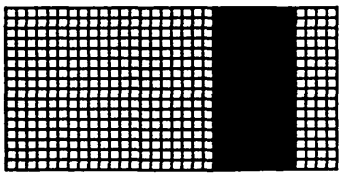
p_1



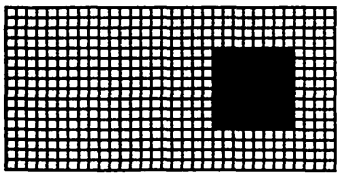
pRC



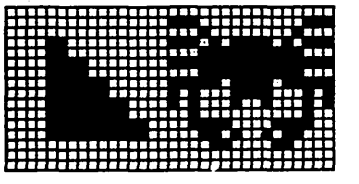
p_V



p_2

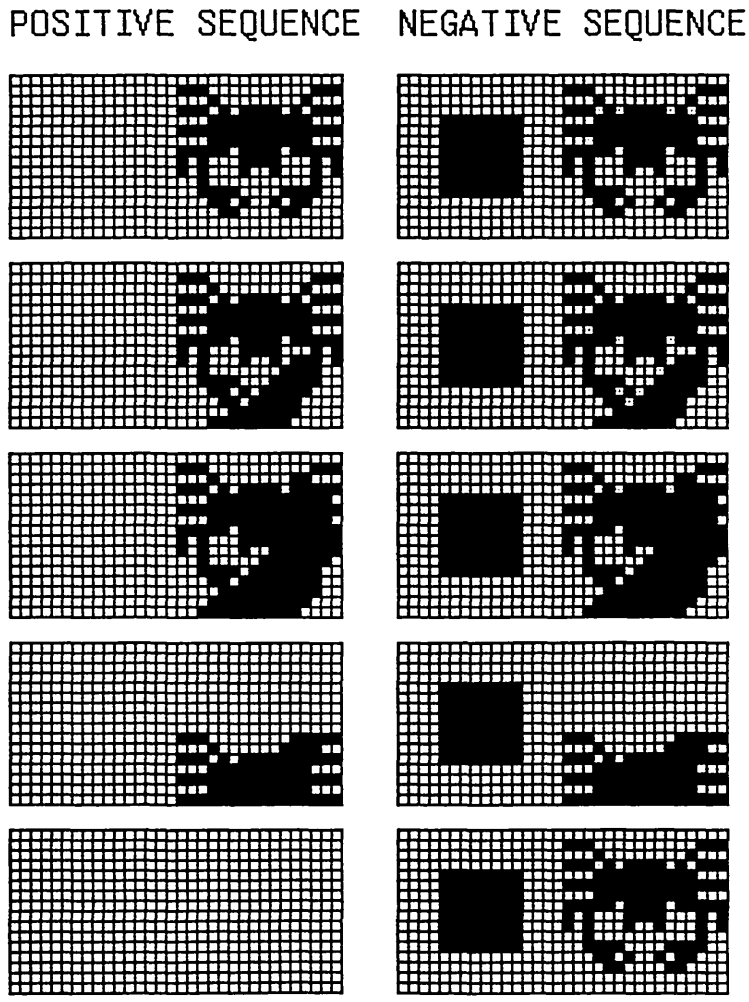


pTC



APPENDIX C

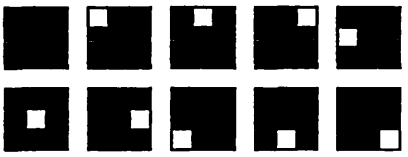
Sample attack sequence with a positive (below left) and negative (below right) stimulus. Reinforcement arrives before presentation of the fifth pattern.



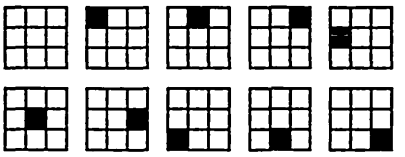
APPENDIX D

The 9-bit input patterns to which each type of classifying cell in OVSIM will output a "1" to signal recognition.

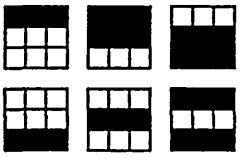
BLACK



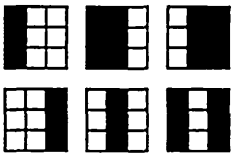
WHITE



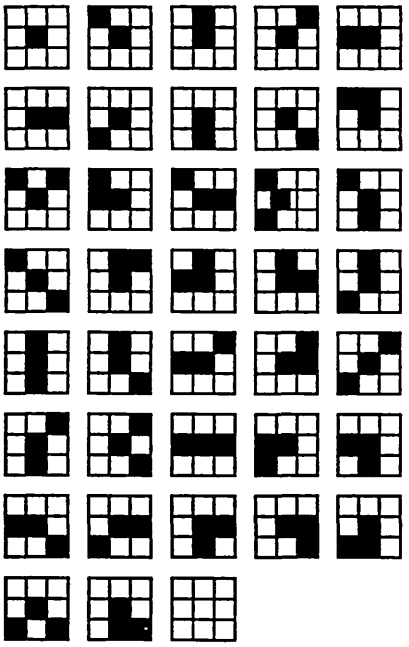
HORIZONTAL EDGES



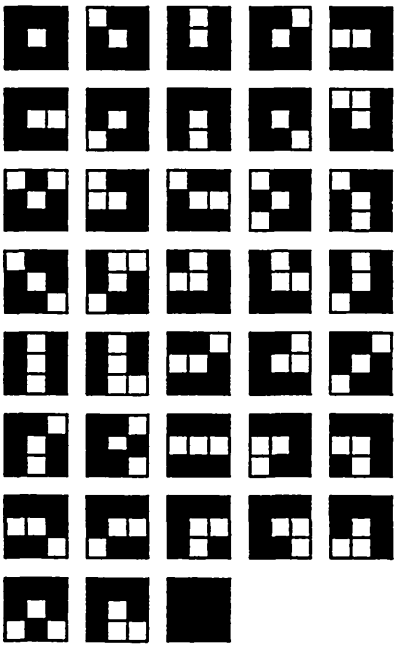
VERTICAL EDGES



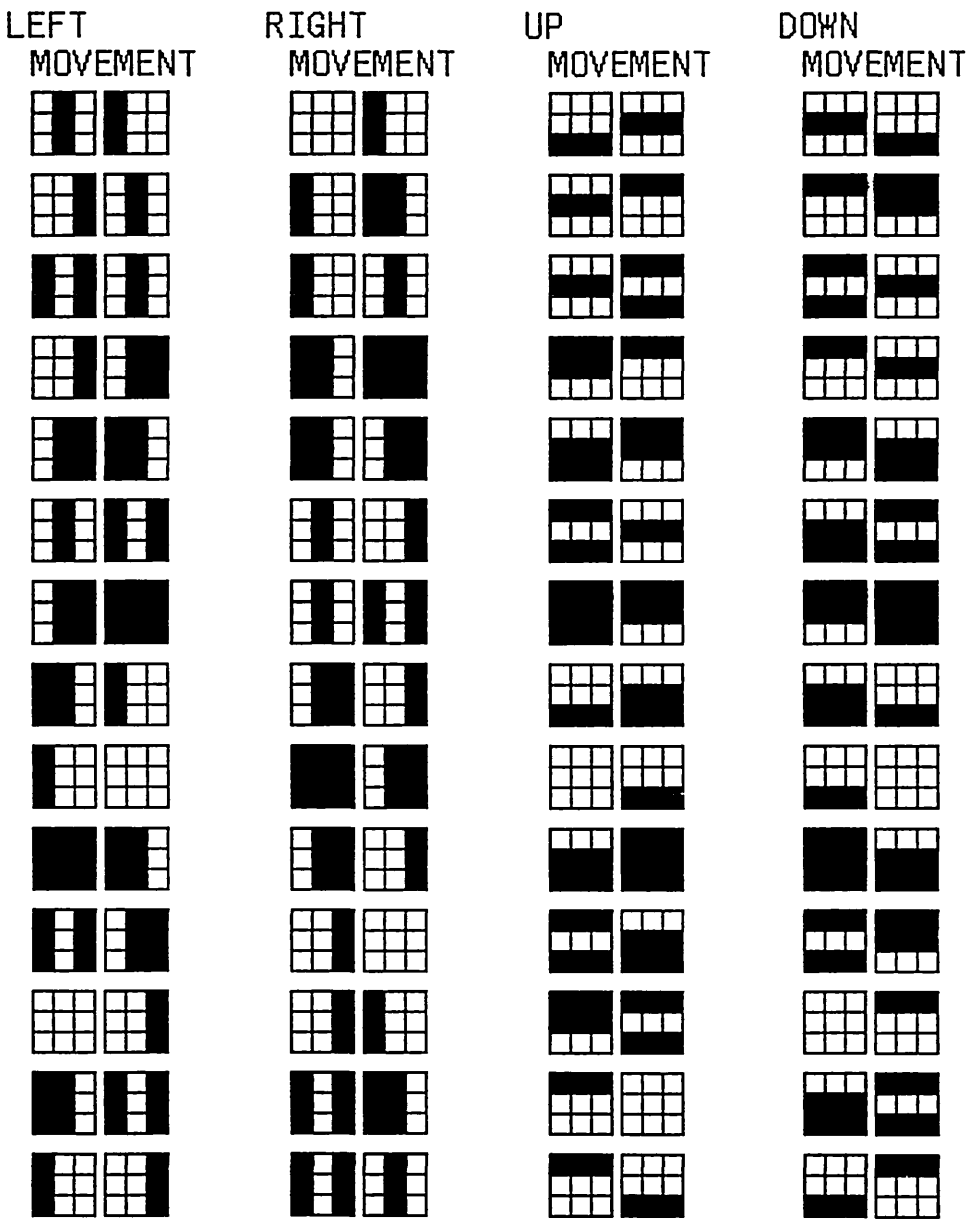
ON-CENTRE



OFF-CENTRE

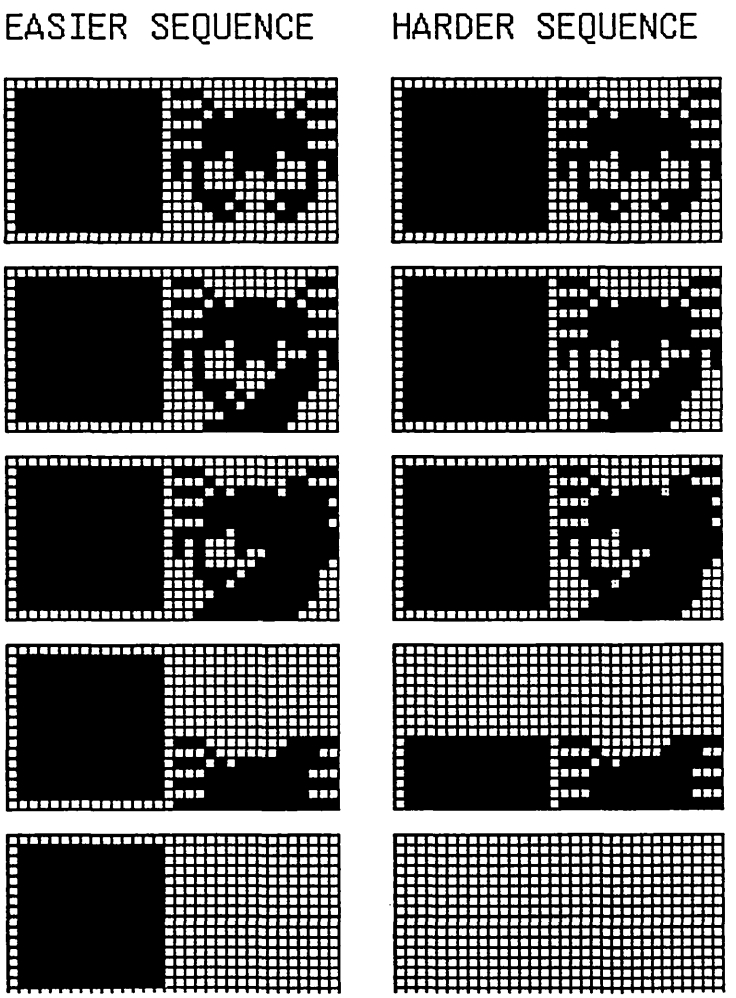


(APPENDIX D, cont.)



APPENDIX E

Two sample attack sequences. Reinforcement arrives after the fourth pattern, making the left sequence "easier" since the current pattern when reinforcement arrives is more like the original stimulus being learned. In the "harder" sequence, the current and orignal patterns are very different, and there can be no transfer of learning about the current pattern to the original stimulus pattern.



APPENDIX F – PUBLISHED PAPERS

Abstracts of INNS 1st Annual Meeting, 1988, p. 205

LEARNING ALGORITHMS FOR PROBABILISTIC NEURAL NETS

by Catherine Myers and Igor Aleksander,
Department of Computing,
Imperial College180 Queen's Gate,London SW7 2BZ England.

Although neural net models show great promise in areas where traditional AI approaches falter, such as pattern recognition, pattern completion and content addressable memory, their success is constrained by slow learning rates and the difficulty of physical implementation; learning strategies such as error-back-propagation are also implausible as biological models.

The Probabilistic Logic Neuron (PLN) [1] [2] represents an attempt to address these issues while retaining the emergent properties of the traditional connectionist models. It is implemented as a variable logic device, with the consequences that it can perform any Boolean function of its inputs, while its use in nets allows drastic reduction in quantity and specificity of connection requirements.

Outputs to the PLN serve as address lines, while the value stored in the bit addressed becomes the output value. Training is thus reduced to the process of storing appropriate binary values into the addresses. This reward/punishment strategy only requires each node to make use of private information and the global error/correct state: there is no need, for example, for the PLN to know to which nodes it is connected or what their states are. Typically, a PLN may store a value which is the probability of outputting a 1 rather than a 0/1 bit itself. This provides for a set of "don't know" states in which 0/1 are output with equal probability.

The typical back-propagation net shows initial goal-directed movement through state space, but quickly resorts to apparently random attempts to approach a nearby global energy minimum. Through use of the "don't know" state to restrict exploration of the state space, and of incremental reward/punishment learning, the PLN net can converge on a solution with minimal wandering. Accordingly, applied to the parity problem and other benchmarks, PLN nets show performance which is orders of magnitude better than back-propagation nets.

Studies are underway to hone the existing PLN learning paradigm to maximise this effect, making best use of the "don't know" information, while retaining the requirement that each node make use only of private local knowledge. The analysis also attempts to define and determine the storage capacity and generalisation ability of these nets.

It is already apparent that the PLN is attractive because of its biologically plausible learning strategy as well as because of its physically realizable structure; PLNs also promise high performance in implementations with regard to other models of connectionist systems.

References

[1] Wing Kay Kan & I. Aleksander, Proc. Ist Int. Conf. on Neural Nets, San Diego, 1987.
[2] I. Aleksander, in Eckmiller & V.D. Marlsberg, Neural Computing, Springer Verlag, 1988.

(APPENDIX F, cont.)

Proc. 1st IEE International Conference on Artificial Neural Networks, London, 1989, pp. 310-314.

OUTPUT FUNCTIONS FOR PROBABILISTIC LOGIC NODES

C. E. Myers

Imperial College of Science, Technology and Medicine, UK

0. ABSTRACT

PLN nets consist of RAM-based nodes which can learn any function of their binary inputs; they require only global error signals during training, and they have been shown to solve problems significantly faster than nets learning by error back-propagation. Output functions for PLNs may be probabilistic, linear or sigmoidal in nature; this paper deals with designing an output function which yields fastest convergence. Experiments with several small problems support the values derived. Choice of an appropriate output function is suggested to be highly problem-dependent, but heuristics for this selection are outlined.

1. THE PLN AND MPLN MODELS

Probabilistic Logic Nodes (PLNs), defined by Aleksander et. al. (1,2), are a class of RAM-based neuron model, designed to combat some of the deficiencies of more ubiquitous weighted-sum-and-threshold approaches; namely that hardware implementation should be straightforward, learning should not be unreasonably slow, and error-correction should require only a global success/failure signal.

Briefly, a PLN is an augmented RAM, in which the I binary inputs to a node form an address into one of the 2^I memory locations, as shown in Figure 1. The output function operates probabilistically on the value stored at this address. During training, an external input pattern is clamped to the input nodes, addressing one location in each; each value accessed is translated into node output and in turn passed as input to other nodes in the net, until external or visible output is produced. If this external output is correct, the value concerned in each node is adjusted so as to increase the probability that its performance will be repeated when next that location is addressed; if the external output is in error, adjustment is made to decrease this probability.

In the simplest case, that of the three-state PLN, every location in every node contains one of three possible values: 0, 1, and "u" – representing "unknown" for the initialised state. When a "u" is accessed, the output function produces a 0 or 1 with equal probability. Stored values are then adjusted from "u" to 0 or 1, as the probability of outputting a 1 is to be encouraged or discouraged, respectively. When an error occurs, as indicated by the external output deviating from the ideal, all currently accessed stored values are reset to "u". Networks of these units are capable of solving hard learning problems (1). A further advantage of the PLN paradigm is that it does not require a specific error signal indicating what the desired output pattern should have been, nor must it calculate errors at each node. A global signal to the effect that error was detected or absent is generated, and all nodes then update on the basis of this general information.

It is possible, however, that the stored values be elements taken from a much larger range than that described above. For example, the values could be selected from the set {0.0, 0.1, 0.2, 0.3, ... 1.0}, representing the probability of outputting a 1 when that value is addressed. Locations in the node might then undergo incremental changes from randomness (0.5) to certainty (0.0 or 1.0). By allowing incremental adjustments, the net learns even more quickly. Myers and Aleksander (3) compared this model to the error back-propagation results given by Rumelhart (4), and found it to be significantly faster and also to generalise well. This learning algorithm also gains the potential for varying the size of the increment in context of how important the trial is. This is however only one possible output strategy for the PLN; the experiments cited in this paper examine the performance of other ranges and interpretations of value.

2. OUTPUT FUNCTIONS

The output function of a node may be described formally as a rule by which the node determines its output, given a certain pattern on its input lines. In the case of the PLN, it is the mechanism whereby stored values are interpreted as affecting the probability that a 1 is output at that node. In equational form, the output function of the three-state PLN is given as:

$$\text{Prob}(\phi = 1) = \xi, \text{ if } \xi \in \{0, 1\}$$
$$= .50, \text{ if } \xi = \text{"}u\text{"} \quad (1)$$

where ξ is the value stored at the location currently accessed and ϕ is the value output. Similarly, for the 11-state PLN described above, we get:

$$\text{Prob}(\phi = 1) = \xi \quad (2)$$

This is a step-function approximating a linear output function. In this case, it is a fairly gross approximation, since there are only eleven steps. A higher number of elements as possible stored values yields a finer approximation, but more bits of RAM are needed in each node to store the 2^I ξ's.

Because of the mutual independence of the ξ's within a single node, the PLN is not restricted to linear functions, and may execute any arbitrary (non-monotonic, non-smooth, non-differentiable, etc.) function. It is true that linear models such as threshold logic (4), ADALINES (5) and Kohonen nets (6) have had considerable success, but there are several reasons why it is desirable to enable the use of non-linearities, particularly the class of semi-linear or "squashing" functions.

As early as 1978, Brodie (7) described cells in *Limulus* eye with a response that was linear about a central range, but which saturated when presented with extreme stimuli: this of course describes a squashing function. Sejnowski (8) reinforces this point in his model of cortex. Biological mimicry is not, however, the only reason to employ sigmoidal output functions: they are essential for signal/noise separation in reverberating nets. Grossberg (9) neatly shows how a sigmoidal output function dissipates noise and quenches units below threshold, while those above threshold are contour-enhanced and may be stored in the reverberating circuit. Thus, non-linearities are a mathematical necessity in complex systems.

The two PLNs described above may be viewed as having maximally different output functions: the first is very "hard" – i.e., there is no intermediate space (except "u") between the conditions of consistently outputting a 0 and a 1. The second, conversely, has a very "soft" limiter, where similar values of ξ yield similar probabilities of outputting a 1. Of course there exist an infinite number of intermediate curves between these two extremes. The current experiments examined effect on speed of learning of PLN output functions varying first in ω, the number of elements representing possible stored values, and second in the softness or hardness of the function which interprets and operates on these values.

3. CHOOSING ω - NUMBER OF POSSIBLE STORED VALUES

A PLN net is said to converge when all locations in all nodes either have probabilities of outputting 1 equal to 0 or 1, or else are never addressed. After convergence, then, any PLN net could be replaced by a set of equivalent three-state PLNs without altering performance.

The advantage of a PLN with many possible stored values during training is that after several reinforcements of a RAM location's value in one direction (e.g., toward outputting a 1), it is very hard to erase that knowledge: in fact, it will take an equal number of negative experiences to return it to "u". Thus it has some protection against errors which may occur in other parts of the net but which affect it as well because the error signal is global and indiscriminate. It will be forced to edge back towards "u", but only one step, and the probability of outputting a 1 need only change by very little in any one cycle. In a network where $\omega = 3$, in contrast, a single error arising anywhere in the net results in one location in every node being reset to "u", and thus knowledge is erased, regardless of whether any individual node was in fact responsible for the error.

A cost of increased ω, besides requiring more RAM, is the difficulty of returning a location to "u" when this is required. Noisy data or an unfortunate ordering of training examples could push a location's

(APPENDIX F, cont.)

value very far from "u" in an undesirable direction, and an equal number of error cycles will be required to reset it.

Ideally, ω must be chosen to balance protection against mistaken erasure versus ability to erase when necessary.

3.1. Definitions

For a given feedforward net of N nodes of fan-in I, with one output node T, describe the current state $S = (S_1, S_2, \cdots S_N)$, and for all nodes i, $S_i = (\xi_{i0}, \xi_{i1}, \cdots \xi_{i2^I})$, where $\xi_{ij} \in R$, the ω-element range of possible stored values. Consider one node i, and let ξ_{ij} be the value stored at the location addressed under the current input j. Then the output of the node, ϕ_i, is given by the output function $\Phi(\xi_{ij}) = Prob(\phi_i = 1)$.

When a net is required to learn a function from input pattern to output pattern, there exist $s \geq 0$ solutions such that in that state the net always gives a correct response. For the current purposes, we consider only problems where $s > 0$.

Then, when the net is in state S, there exist one or more "closest" solutions S_c in function space, where the difference function is at a minimum:

$$f = \sum_{i=1}^{N} \sum_{j=0}^{2^I-1} abs(\Phi(\xi_{ij}^{S}) - \Phi(\xi_{ij}^{S_c})) \qquad (3)$$

Given one S_c, call the output from each node in that state ϕ_i^*. Then we may define each ξ_{ij} to be "Right" if when addressed,

$$sgn[Prob(\phi_i = 1) - .5] = sgn[Prob(\phi_i^* = 1) - .5] \qquad (4)$$

and "Wrong" otherwise. More generally, we may say that a node outputs the Right/Wrong value with probability R_ξ/W_ξ when a given ξ is addressed. Usually, these probabilities can only be measured at the output node, T.

Then the problem reduces to a choice of ω such that there is a high probability of erasing Wrong values even when $R_T > W_T$, and simultaneously a low probability of erasing Right values even when $W_T > R_T$.

3.2. Choosing low ω to allow desired erasure

A RAM location q in node i is set at $\xi \in R$. At the current net state, P patterns address it; for P^R, $\phi_T = \phi_T^*$ while for P^W, $\phi_T \neq \phi_T^*$. Notice that it is possible that $|P^W| > |P^R|$, even if ξ is Right, or for $|P^R| > |P^W|$ when ξ is not Right, depending on activity elsewhere in the net.

In order to return ξ to random, so it might be reset, there must occur y trials such that the number of elements of P^W appearing exceeds the number of elements of P^R by at least $\left\lfloor \frac{\omega}{2} \right\rfloor$, as in the worst case, ξ is $\left\lfloor \frac{\omega}{2} \right\rfloor$ steps away from "u". For simplicity, we consider only the $x < y$ trials during which ξ is actually accessed, and assume the rest of the net is subject only to minor change, so that S_c does not change.

If clamped training inputs are selected randomly, elements of P^R and P^W occur as Bernoulli trials. So the probability that at least m elements of P^W occur in x trials is:

$$Prob(X \geq m) = B(x,m) + B(x,m+1) + \cdots + B(x,x) \qquad (5)$$

where $B(u,v) = \binom{u}{v} W^v R^{u-v}$, $W = \frac{|P^W|}{|P|}$, $R = \frac{|P^R|}{|P|}$, and $x \geq m$. But this is not strict enough: it does not suffice that m events occur unless

$$m \geq (x-m) + \left\lfloor \frac{\omega}{2} \right\rfloor, \text{ or } 2m - x \geq \left\lfloor \frac{\omega}{2} \right\rfloor \qquad (6)$$

So, the probability that enough elements of $|P^W|$ appear in x trials to reset ξ to randomness is given by:

$$P^1 = Prob(\xi \text{ reset within } x \text{ trials}) = \sum_j \sum_k B(j,k) \qquad (7)$$

where j ranges from $\left\lfloor \frac{\omega}{2} \right\rfloor$ to x, k ranges from $\frac{1}{2}\left(\left\lfloor \frac{\omega}{2} \right\rfloor + j\right)$ to j, and $\sum$ summates increasing its index by 2.

3.3. Choosing high ω to avoid mistaken erasure

This time, to avoid ξ being reset to random, there must not occur y trials (of which some x are relevant) such that the number of patterns accessing ξ and generating $\phi_T \neq \phi_T^*$ exceeds those generating $\phi_T = \phi_T^*$ by at least $\left\lfloor \frac{\omega}{2} \right\rfloor$. The probability that this does not occur is:

$$P^2 = Prob(\xi \text{ not reset in } x \text{ trials}) = 1 - P^1 \qquad (8)$$

3.4. A compromise

The desired solution maximises both P^1 and P^2: if we consider a sum-of-squares function,

$$(P^1)^2 + (P^2)^2 = P^{1^2} + (1-P^1)^2 = 1 - 2P^1 + 2(P^1)^2 \qquad (9)$$

a maximum occurs at $P^1 = +\frac{1}{2}$, i.e., where $P^1 = P^2 = \frac{1}{2}$. Therefore, the equation to solve is

$$f(\omega) = \sum_j \sum_k B(j,k) \approx \frac{1}{2} \qquad (10)$$

Notice that this equation depends on R_ξ, W_ξ, and ω, but not on N, I, or the depth of the net.

Figure 2 shows optimal values of ω, such that (10) is satisfied, as a function of P^W for sample values of x. Ideally, the net should be constructed to minimise ω (and hence the amount of RAM needed) and also x (and hence convergence time). If it is reasonable to assume that P^W deviates around a mean of 0.5, it is seen from Figure 2 that for small x, optimal ω clusters in a range $5 \leq \omega \leq 15$.

Experimental data were gathered to check this assumption and derived ω. Figure 3 shows speed of convergence for small nets of 3 and 7 nodes learning the parity problem at different ω. In the case of the smallest net, $\omega = 11$ led to fastest convergence, while in the larger case a smaller ω performed somewhat better, although for about 75% of the nets, convergence occurred in roughly the same time for all values of ω.

Finally, the nets were set a task involving generalisation. The problem was to recognise a patch of 3 or more 1s in a five-bit input string, including the cases where the patch had moved partially offscreen but at least two of its bits showed. The negative patterns were instances of single patches of 1 or 0 bits, or patches of 2 bits centered – and thus not tips of a larger patch. Figure 4 shows the training set and learning speeds. Nets with $\omega = 11$ learned the distinction fastest, with 76% of sample nets converging in less than 100 training cycles; the $\omega = 21$ nets lagged well behind the others, and in fact 54% failed to converge within 5000 cycles.

After each net had been trained, it was required to respond to all of the $2^5 = 32$ possible inputs. Of the 16 unseen patterns, 7 were examples of positive patterns distorted by one bit; 9 were negative ones with one bit of noise. Of the total 32 patterns, $\omega = 6$ nets averaged 27.42 correct responses; $\omega = 11$ nets averaged 27.45 correct, and $\omega = 21$ averaged 28.30 right. In all cases a reasonable degree of generalisation was achieved. However, the $\omega = 21$ nets showed the most capability to adapt their responses to new stimuli. Because in nearly half the cases the $\omega = 21$ nets had failed to converge, some nodes i in the net had locations such that, when accessed, $0 < Prob(\phi_i = 1) < 1$. Therefore, there was a possibility of outputting different values at these nodes, even given the same address. This flexibility is responsible for the nets' ability to generate a wider range of input in response to new patterns, and hence results in a slightly higher percent correct on the generalisation task.

Overall, empirical data tend to support the theoretical prediction: that $5 \leq \omega \leq 15$, and in specific, $\omega \approx 11$, should lead to fastest convergence.

For the remainder of this paper, ω will be held constant to the value of 11.

4. CHOOSING Φ - THE OUTPUT FUNCTION

The second main concern of this paper is selection of Φ, the output function. A node i addresses location q which stores some value $\xi \in R$. Then its output ϕ_i is determined by:

$$\Phi(\xi) = Prob(\phi_i = 1) \qquad (11)$$

Φ may range from relations which approximate linear functions to ones which approximate threshold functions.

4.1. Derivation of Φ

Consider again a feedforward tree, with one output node, T. Define R_ξ as the Right value of ξ with respect to a nearest solution S_c, and define ϕ_i^* as the Right output from node i under the current input. ξ is adjusted toward R_ξ when $\phi_i = R_\xi$ and $\phi_T = \phi_T^*$, and thus repetition of the event is encouraged. Similarly, when $\phi_i \neq R_\xi$ and $\phi_T \neq \phi_T^*$, the current event is made less likely to reoccur, and the desired one encouraged. If either of the other two events occur ($\phi_i = R_\xi$ and $\phi_T \neq \phi_T^*$, or $\phi_i \neq R_\xi$ and $\phi_T = \phi_T^*$), ξ is trained wrongly – i.e., away

(APPENDIX F, cont.)

from R_ξ. It is therefore desirable to select Φ such that for a value ξ addressed in node i:

$$Prob(\phi_i = R_\xi \cap \phi_T = \phi_T^*) + Prob(\phi_i \neq R_\xi \cap \phi_T \neq \phi_T^*)$$
$$>> Prob(\phi_i = R_\xi \cap \phi_T \neq \phi_T^*) + Prob(\phi_i \neq R_\xi \cap \phi_T = \phi_T^*) \quad (12)$$

Assume $R_\xi = 1$ and $\phi_T^* = 1$; the other cases are of course symmetric. Then (12) reduces to:

$$\Phi(\xi)\sum_v Prob(v\ accessed|\phi_i = R_\xi)\Phi(v))$$
$$+ (1-\Phi(\xi))\sum_v Prob(v\ accessed|\phi_i \neq R_\xi)(1-\Phi(v))$$
$$>> \Phi(\xi)\sum_v Prob(v\ accessed|\phi_i = R_\xi)(1-\Phi(v))$$
$$+ (1-\Phi(\xi))\sum_v Prob(v\ accessed|\phi_i \neq R_\xi)\Phi(v) \quad (13)$$

where v is a value stored in some location in T, and $Prob(v accessed)$ represents the likelihood that the location containing v is addressed given the current output from node i. $Prob(v\ accessed|\phi_i = R_\xi)$ is of course independent of $\Phi(v)$ and $\Phi(\xi)$ This simplifies to:

$$\Phi(\xi)\sum_v Prob(v\ accessed|\phi_i = R_\xi)\left[2\Phi(v)-1\right]$$
$$>> (1-\Phi(\xi))\sum_v Prob(v\ accessed|\phi_i \neq R_\xi)\left[2\Phi(v)-1\right] \quad (14)$$

A general form for Φ is $(1+e^{-x})^{-1}$. We consider a particular instantiation, $\Phi(x) = (1+e^{\alpha(-2x+1)})^{-1}$ – allowing x to range from 0 to 1 instead of the more usual -1 to +1. The topic of concern is then to select α to maximise inequality (15).

$$\frac{1}{1+e^{\alpha(-2\xi+1)}}\sum_v Prob(v\ accessed|\phi_i = R_\xi)\left[2\Phi(v)-1\right]$$
$$>> \frac{e^{\alpha(-2\xi+1)}}{1+e^{\alpha(-2\xi+1)}}\sum_v Prob(v\ accessed|\phi_i \neq R_\xi)\left[2\Phi(v)-1\right] \quad (15)$$

or,

$$\frac{\sum_v Prob(v\ accessed|\phi_i = R_\xi)\left[2\Phi(v)-1\right]}{\sum_v Prob(v\ accessed|\phi_i \neq R_\xi)\left[2\Phi(v)-1\right]} >> e^{\alpha(-2\xi+1)} \quad (16)$$

The right side of this equation is a constant for a given value of ξ, and has a minimum as $\xi \to 1$, depending on the assumption $R_\xi = 1$. The left side, however, will change dramatically as the net is trained, since it depends on the probability of accessing different locations in the top node given an output from i. Therefore, the best solution to this equation is to make α as large as possible, to maximise the frequency with which states in the net will satisfy the inequality. The result is an output function, Φ, which should be made to resemble a very steep sigmoidal curve.

The result is intuitively satisfying: it suggests that once a node location is 'committed' to an output, i.e., that it has been reinforced even once away from randomness and toward either 1 or 0, it should output that value consistently. This allows other locations in the net to organise around one another with some confidence that all are behaving as they expect to behave when fully trained.

4.2. Experimental Results

Five limiters were compared in performance: approximating

$$\Phi(x) = \frac{1}{1+e^{\alpha(-2x+1)}} \quad (17)$$

where $\alpha \in \{2.5, 4, 5, 10, 25\}$ and $x \in \{0, 0.1, 0.2, \ldots 1.0\}$. The correspondences between the continuous functions and the Φ approximations are shown in Figure 5.

One hundred nets were trained with each value of α for the two parity problems described in the previous section; in both tasks, there was a clear tendency for nets with hardest limiters to converge fastest and those with softer limiters to converge more slowly. Figure 6 shows these results, and the mean and median time to convergence for each type of net is given in Table 4a.

Set the generalisation task described above, all 5 categories of net tended to learn to recognise the training set in approximately the same time, with 50% of all trials resulting in convergence within 1500 pattern presentations. Out of the 32 patterns in the generalisation set, nets with $\alpha = 2.5$ responded correctly to an average 27.45 patterns; those with $\alpha = 4$ to an average 27.58 patterns, and those with $\alpha = 5$, 10 and 25 to averages of 27.26, 27.32 and 27.20 patterns, respectively. These success rates are all very similar, as expected, since after convergence there should be no difference between nets using any particular limiter in their ability to execute an associative function. As all classes of net converged in similar time, and therefore all had a similar number of trials unconverged at the end of 5000 pattern presentations, no one class of net showed the advantage in generalisation of the unconverged nodes in the $\omega = 21$ nets of the previous section.

These results support the theory that a hard limiter, for example a high value of α in the functions considered here, leads to faster convergence, while having little effect on performance after training is completed.

5. CONCLUSIONS

Several assumptions are implicit in the results presented here, and they warrant restatement. The nets considered are feedforward pyramids, being trained on problems for which a solution exists, via a training schedule which involves a random ordering of training patterns. This is a constrained class of topology and task, but one which is still quite powerful.

Given these assumptions, an MPLN net may be designed which will tend to converge as fast as possible: namely, its nodes contain stored values selected from a range of $5 \leq \omega \leq 15$ elements, and interpreted according to a threshold-like output function. The experiments described to support these claims are small both in terms of the number of nodes and also in terms of size of state space relative to number of solutions available. They are useful, however, since a small number of distinguishable solutions exist and since the parity problems are arguably the "hardest" of the hard learning problems.

It is not the case that the ω and Φ defined here are universally optimal; it is not clear in the first place that speed of convergence is a necessary criterion to judge the "success" of a network – although it is probably the most frequent. There are occasions when a soft limiter, for example, will be desirable despite its slowness. One obvious example involves a state space with abundant and deep local minima, where probabilistic noisy outputs are necessary; in effect, a net using a steep output function forms quick and binding opinions, whereas a net with a more linear output function makes conservative ones, which still allow occasional lapses into the opposite output. This ability would prove important in simulation of an automata existing in a changing environment, where convergence per se is not possible, and where a net might be more successful if some of its nodes, say, output a one 75% of the time, and occasionally output a zero to test the effects in the current environment.

Appropriate choice of parameters is therefore highly dependent on the size, shape and complexity of the problem space, and also causes subtle changes in the way the net organises to solve the problem – particularly in the speed with which nodes commit to a particular output in response to an input pattern. The values of ω and Φ presented here cannot therefore be purported to be optimal under all conditions, merely as especially useful and as good first approximations for later fine-tuning, as necessary.

TABLE 1 - Performance data for experiments on learning 4-bit and 7-bit parity. Mean (with standard deviation) and median times to convergence are shown, along with time for 90% and 100% of nets tested to converge.

3-Bit Parity

α	mean	st.dev.	median	90%	100%
2.5	241	132K	150	500	3000
4	165	21K	150	400	900
5	186	30K	150	450	1000
10	123	15K	70	300	700
25	81	9K	70	200	705

7-Bit Parity

α	mean	st.dev.	median	90%	100%
2.5	6.8K	46K	5.0K	14K	35K
4	8.9K	66K	6.5K	25K	45K
5	7.2K	45K	5.0K	15K	35K
10	4.8K	28K	3.5K	11K	35K
25	3.7K	28K	2.5K	9K	25K

(APPENDIX F, cont.)

Acknowledgements

I am grateful to Igor Aleksander and the Neural Systems Engineering Group for encouragement and useful discussions. This research was supported by the NSF (USA).

6. REFERENCES

1. Aleksander, I., 1988, in Eckmiller, R and von der Malsburg, C., eds., "Neural Computers," Springer-Verlag, Berlin, 189-197.
2. Kan, W. and Aleksander, I., 1988, in IEEE Proceedings International Conference on Neural Networks, San Diego, 541-548.
3. Myers, C. and Aleksander, I., 1988, in Proceedings First INNS Annual Meeting, Boston.
4. Rumelhart, D., Hinton, G., and Williams, R., 1986, in Rumelhart, D. and McClelland, J., eds., "Parallel Distributed Processing: Explorations in the Microstructure of Cognition", vol. 1, MIT Press, London, 318-362.
5. Widrow, B. and Winter, R., 1988, Computer, 21, 25-39.
6. Kohonen, T., 1984, "Self-Organisation and Associative Memory", Springer-Verlag, New York.
7. Brodie, S., Knight, B. and Ratliff, F., 1978, J. Gen. Psychol., 72, 129-154, 162-166.
8. Sejnowski, T., 1981, in Hinton, G. and Anderson, J., eds, "Parallel Models of Associative Memory", Lawrence Erlbaum, Hillsdale, NJ, 189-212.
9. Grossberg, S., 1973, Studies in Appl. Math., 50, 213-257.

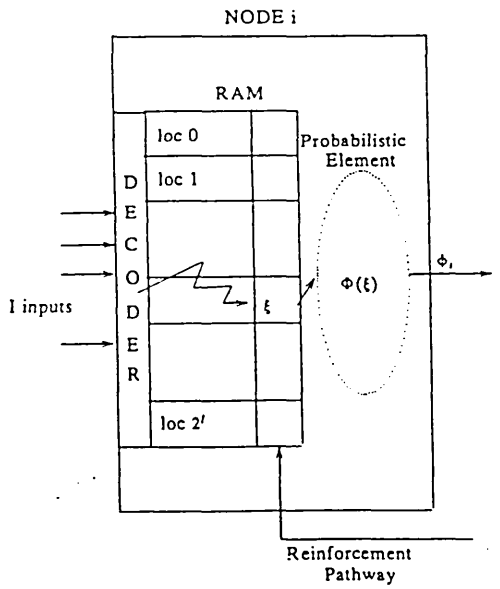
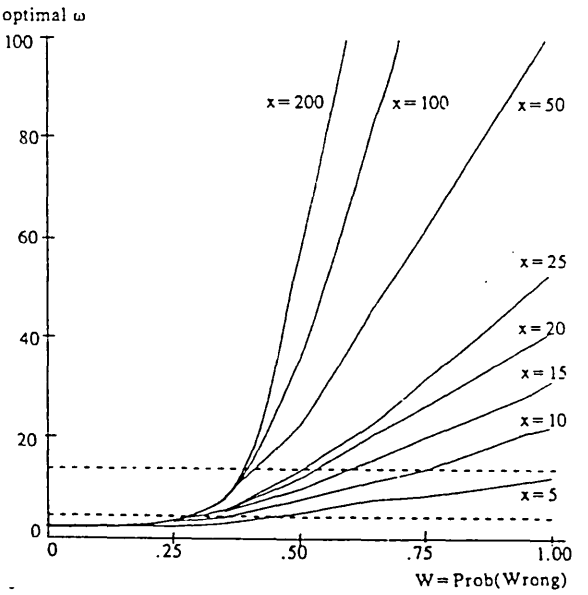


Figure 1 The PLN; I binary inputs address a location in RAM memory; the value accessed, ξ, is passed to the probabilistic output function, Φ, which converts it to binary output, ϕ_1.



Dashed line indicates region of $5 < \omega < 15$.

Figure 2 The optimal values of ω, as given by Equation (10), as a function of P^w; plotted for several values of x, the number of training cycles.

(APPENDIX F, cont.)

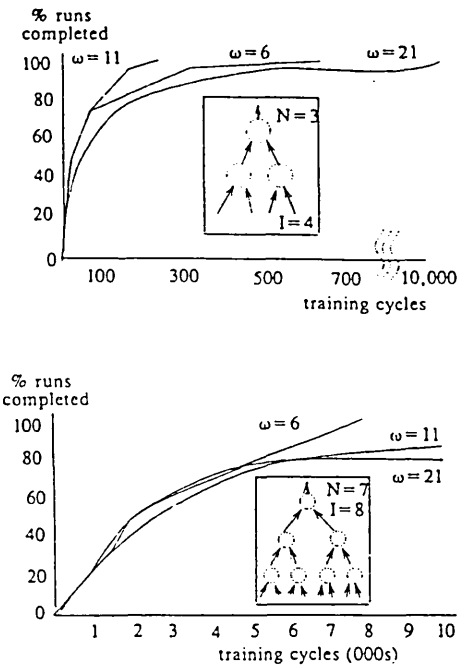


Figure 3 Speed of convergence for two parity problems, for several values of ω. Net topologies are shown in insets; sample size is 100 nets for each ω.

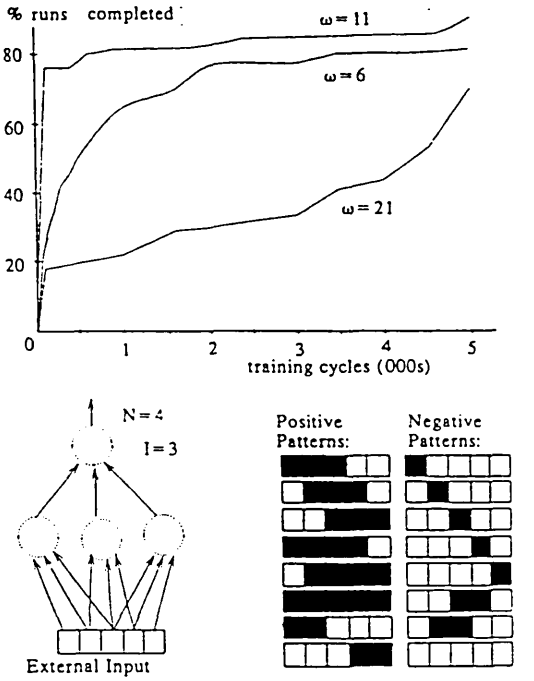


Figure 4 Speed of convergence on the generalisation task for several ω. Net toplology and the 16 training patterns are shown; sample size is 100 nets for each ω.

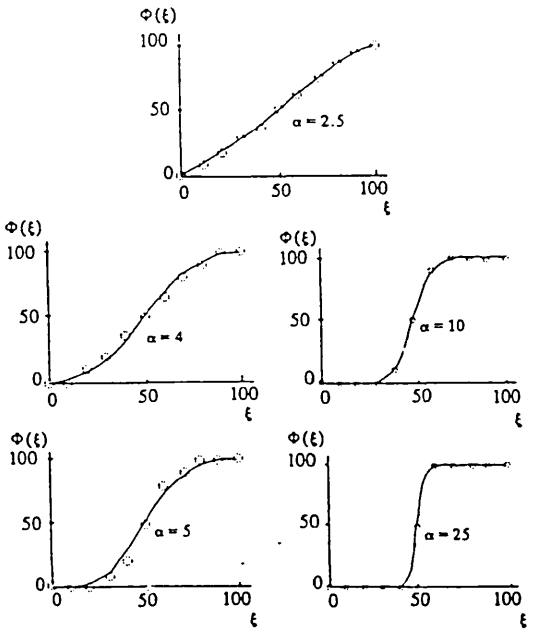


Figure 5 The 5 Φ tested. Solid line is $1/1+e^{\alpha(-2x+1)}$; open circles are actual probabilities used for each value of ξ.

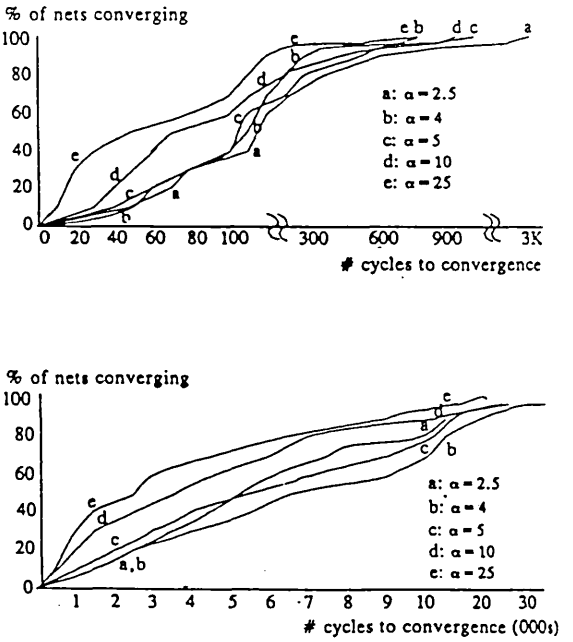


Figure 6 Speed of convergence for two parity problems, for several Φ. Topologies as in Figure 3; sample size is 100 nets for each Φ.

Proc. INNC-90-PARIS, Paris, 1990, pp. 860-863.
(APPENDIX F, cont.)

REINFORCEMENT LEARNING WHEN RESULTS ARE DELAYED AND INTERLEAVED IN TIME

Catherine Myers
Neural Systems Engineering
Department of Electrical Engineering
Imperial College, London SW7 2BT ENGLAND

Many real-world problems involve sequences where a automaton executes an action but there is some delay before the results of that action become apparent. A system is presented which learns to associate early stimuli with later reinforcement by buffering unfamiliar input images until that reinforcement arrives. It is shown to learn to predict the immediate results of various actions in a given state, to avoid entering negative next-states, and also to avoid entering positive next-states which lead in turn only to negative states. The system is capable of learning across indefinitely long reinforcement delays while only buffering a small number of past states locally at the nodes.

Introduction. In the physical world, most events which entail (positive or negative) reinforcement occur some time before the results actually arrive. For example, when an animal sees a food-like image and decides to approach, grasp and ingest the object, the positive reinforcement (taste) does not arrive until the end of the sequence. The original visual image of the food is supplanted by a series of intervening ones – the final ones do not even include the food image as it is out of sight inside the mouth. Yet even very simple animals learn to bridge this time gap and associate distal images with appropriate approach responses.

Solving this problem requires two abilities: first, that the memory of earlier images be available when the reinforcement arrives; second, that learning be possible even though this reinforcement is only an estimate of "goodness" rather than a full desired output as is traditionally provided for supervised learning in neural networks.

The latter issue, reinforcement learning, has received some attention, notably from Widrow [1,2], Barto and Sutton [3], Klopf [4] and Aleksander [5]. A variant of Aleksander's model, the MPLN (Multi-valued probabilistic logic node) [6] is used here.

The question of learning with delayed reinforcement is often approached in one of two ways. One solution is to maintain a buffer of all previous states, possibly each with an eligibility that decays with time, and then to update each according to its eligibility when reinforcement arrives. This quickly becomes impractical as the number of possible states grows. A second solution is to buffer only the S states immediately previous to the current; but this precludes the system from learning about images which occur S+1 time steps before their associated rewards.

The system investigated here also maintains a buffer of some S previous states, but these are not necessarily those that occurred in the immediately preceding time steps. Rather, they are the S most unfamiliar previous stimuli. A state is placed in the buffer if its outcome is more unpredictable than some item currently in storage, and it overwrites that item. Simultaneously, the longer an item has been in the buffer, the more likely it is to be ousted by a new item. If, on the other hand, the effects of the new stimulus are predictable with great certainty, it is unnecessary that the item enter the buffer – since there is nothing new to be learned about it.

In this way, the system can keep a small number of previous states available, and yet learn to associate reinforcement with states which occurred indefinitely earlier.

The Task. The problem considered here is based loosely on the idea of an automaton learning to select food. A set of M types of element (each a 64-bit pattern) exist in the world; of these some $P \subset M$ are positive while the remainder $N \subset M$ are negative. At each time cycle, the automaton is in some state $x \in M$, and has a choice of moving left, right or straight ahead; a transition matrix determines the next state: $f(x, \text{move}) \rightarrow y \in M$. If $y \in P$, a positive reinforcement is supplied immediately (as if the automaton experienced the taste of food). If the automaton enters $y \in N$, there is an immediate positive reinforcement, followed by a strong negative reinforcement delayed by some d time steps (as if there was a taste of food later followed by nausea).

Thus the task is to learn to predict the three adjoining states from the current one, and also to select the moves which result in some $y \in P$ and not $y \in N$, even though the results are delayed and contradictory signals may intervene. For example, negative reinforcement may not arrive until some time after $x \in N$ has been entered, and it may arrive just after some element of P has been entered.

The Model. The adaptive system consists of three basic parts: the Associator Module (AM) which, given the current input image and a suggested next move, predicts the resulting next state; the Judge Module (JM) which, given a predicted next state, estimates the desirability of entering that state; and the short-term store (STS) which stores recently entered states in readiness for the arrival of results. The complete system is shown in Figure 1. The move selected will be the one which results in a prediction from AM to which the JM responds most highly.

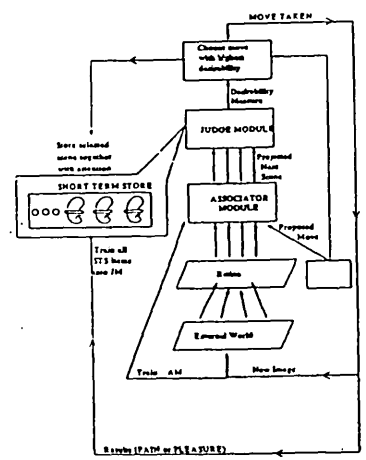


Figure 1. The system for learning with delayed reinforcement. The AM predicts next states for given moves, the JM judges desirability of next states, and move is chosen accordingly. Input scenes are stored in STS buffer until reinforcement arrives, whereupon judge is trained to adjust desirability predictions.

There is a limit, S, on the number of elements which may reside in STS at any one time. As each element enters STS, it is assigned a certain strength or attention which decays with time. When a new element is to be stored in STS, it overwrites the resident element with the weakest attention. Initial attention, in the simplest case, is a constant larger than the longest possible delay D: STS then reduces to an S-previous-element buffer of the sort discussed above.

The AM consisted of MPLN trees: each containing 8 10-input MPLNs feeding into an 8-input top node. The lower level nodes each sampled 8 input bits (randomly, but all 64 input bits were used) plus two bits encoding the move to be evaluated. 64 of these trees existed, each outputting a bit, so that the predicted next state could be

reproduced. Two JMs were investigated: a multi-layer version, MLJM, of 8 MPLN trees – each similar to the AM trees, but without the move information; the total responses from all top nodes gave a score 0..8 of how desirable the projected state was judged to be; and a single-layer version, SLJM, consisting simply of 25 8-input MPLNs, sampling the input retina randomly but evenly, and outputting total response in the range 0..25.

When an action is selected, the current state is stored in STS, along with up to 4 other previous states (S=5); this is done locally at each node, and so could be synchronous. When reinforcements arrive, each STS pattern is reapplied to the JM, which is then trained by a standard MPLN learning algorithm [6] to increase or decrease its desirability measure for those patterns. The AM is updated after each cycle, by the same algorithm, to produce the received next state of the world in response to the previous state and move taken.

Results. The AM, trained alone to produce next state from current state plus move, predicted with 90% accuracy (measured in bits right) after 300 passes through the pattern set; within 900 passes it achieved 99% accuracy, and took 6,000 passes to perform perfectly.

The MLJM, also trained separately, learned within 3,000 cycles (~100 passes through its training corpus) to respond strongly to all elements of P and weakly to all elements of N. This module was then paired with the trained AM, so that temporal effects (such as "avoid positive states which lead only to negative ones (cul-de-sacs)") came into effect. Within a further 2,000 cycles, the complete system learned all the necessary associations: including avoidance of the state which led only to elements of N, but non-avoidance of states which led to an element of N but also at least one element of P. However, by the conclusion of this training, the system had experienced over 60 negative reinforcements or an average of 20 per element of N – low by neural engineering standards, perhaps, but excessive when compared with animal learning.

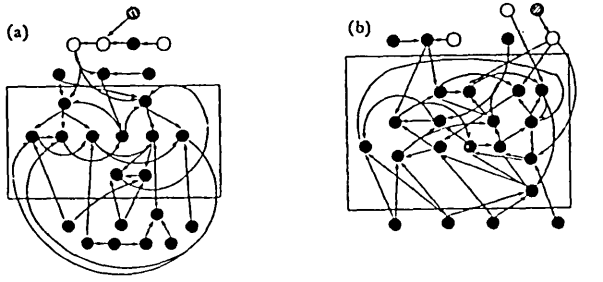
Training SLJMs in conjunction with the pretrained AM, the system tended to receive an average of 5.5 negative reinforcements within the first 500-2,000 cycles, after which it would never again enter a negative state. This is clearly much faster learning than was obtained with the MLJM; however, the system did not learn so comprehensively. One solution found is shown in Figure 2a. Not every positive state is re-entrant – in fact, rather less than half are. This is still a valid solution, since the system receives positive reinforcement on every cycle. Because the system learns so fast, some positive states are entered only once or twice before the system settles on a stable behaviour and may not be entered again.

Simultaneous training of AM and SLJM resulted in more negative reinforcements during training, as early output from the AM was nearly random and therefore useless to the SLJM. The average number of negative reinforcements was 24, still considerably less than with the MLJM even trained alone. Typically, most of these occurred within the first 500 cycles. Solutions found by these systems included more re-entrant states than when the AM was pre-trained; one solution, shown in Figure 2b, has 15 of the 22 positive states re-entrant, with all negative elements and the cul-de-sac transient and hence never re-entered.

The observed tradeoff is that the MLJM solves the problem perfectly, while the SLJM finds acceptable approximations to the solution within much fewer negative experiences. The advantage of using multi-layer systems is their ability to use hidden nodes to form internal representations. In this task, such representations are unnecessary, and the SLJM was perfectly adequate – particularly as, composed of MPLNs, each node could learn any boolean function of its inputs.

Conclusions. A system has been described to learn under conditions of a global scalar reinforcement signal which arrives with some delay; it achieves this by keeping a store of S recent inputs with unpredictable outcomes. This store may be local to the nodes, allowing parallel updates. The system has been shown capable of making second order temporal predictions, such as "avoid a state which is itself positive but which leads inevitably to a negative one." The results described here involve a buffer size which is at least as large as the maximum possible delay; a future paper will show the system capable of learning even when the maximum delay is longer.

Figure 2. Example solutions found by SLJM with pre-trained AM (a) and by SLJM and AM trained together (b). Filled circles = positive states, white circles = negative states, striped circle = cul-de-sac state. Box enclosed re-entrant states.



References

[1] Widrow, B., Gupta, N. and Maitra, S. Punish/reward: learning with a critic in adaptive systems. *IEEE Trans. on Systems, Man and Cybernetics*, SMC-3(5), 455-465, 1973.

[2] Widrow, B. and Smith, F. Pattern-recognising control systems. *Computer and Information Sciences*, Eds. J. Tou and R. Wilcox. Washington, D.C.: Spartan Books, 1964, pp. 288-317.

[3] Barto, A., Sutton, R. and Anderson, C. Neuronlike adaptive elements that can learn to solve difficult learning control problems. *IEEE Trans. on Systems, Man and Cybernetics*, SMC-13(5), 834-851, 1983.

[4] Klopf, A. H. A neuronal model of classical conditioning. *Psychobiology*, 16(2), 85-125, 1988.

[5] Aleksander, I. Logical connectionist systems. *Neural Computers*, Eds. R. Eckmiller and C. von der Malsburg. Berlin: Springer-Verlag, 1988, pp. 189-197.

[6] Myers, C. Output functions for probabilistic logic nodes. *Proc. First IEE International Conf. Artificial Neural Networks*, London, October, 1989, pp. 310-314.