Weighted Nodes and RAM-Nets: A Unified Approach

K.N. GURNEY

Department of Human Sciences, Brunel University, Uxbridge, Middx., United Kingdom

CONTENTS Page

Abstract

A general model of so-called 'RAM-based nodes' which uses continuous sites values on the hypercube and processes analogue inputs is developed. A closed expression is given for the node's functionality and is shown to be equivalent to that for the sigma-pi nodes described by Rumelhart and McClleland (1986) of which, the customary semilinear units or TLUs are a subset. This is one of several points of contact developed between RAM-based nodes and those that use sums of weights of input terms. Points of contrast are also discussed in the context of generalisation, training and implementation. It is shown that the models may be instantiated with time stochastic processing and RAM implementations realised by quantising the site values.

Key words: Neural Nets, Sigma-pi, Higher order, RAM nets, Hypercubes

1. Introduction

Networks of arbitrary Boolean functions (Aleksander, 1973; Aleksander & Stonham, 1979; Aleksander, Thomas & Bowden, 1984; Martland, 1987; Milligan, 1988; Austin, 1987) and, more recently Probabilistic Logic Nodes (Myers & Aleksander, 1988; Myers, 1989) and pRAMS (Gorse & Taylor, 1990) have traditionally been viewed as a radical alternative to the use of nodes whose activation is defined by a weighted sum of inputs. Part of the motivation for their use is that they readily lend themselves to implementation in RAM technology, leading to their collective title, 'RAM-based node' and their networks as 'RAM-nets'. This has placed them somewhat outside the mainstream of Neural Net research which has usually dealt with McCulloch and

Pitts type neurons (TLUs), or semilinear nodes which have a sigmoidal output function. Among the differences are that the latter can deal with analogue inputs and may have their functionality expressed in a closed analytic form, allowing proof of learning convergence in a supervised regime.

It is shown in this paper that the RAM-nets may be thought of as having both these properties. In particular, the classic training schemes using Reward Penalty (Barto & Jordan, 1987) and Back-propagation (Rumelhart & McClelland, 1986) may be applied. The route to this uses models with continuous (later quantised) values at each address and time stochastic processing of analogue inputs. There is a closed form for the functionality which may be expressed in a way equivalent to that for the sigma-pi units described by Rumelhart & McClelland (1986). These *higher order* nodes have, of course as a subset, the conventional semilinear units.

The models used are not bound to any hardware implementation but may still be built using RAM devices. The nodes are considered to be defined by a population of values at the vetices or sites of a hypercube and will be referred to as cube-based or *cubic*. The process of training set generalisation may be thought of as related to the clustering of similar site values with respect to the hypercube topology.

One variant of this scheme is the Multi-Cube-Unit (MCU) which sums the activation from several cubes before producing output. This constrains the functionality and is the closest equivalent to the TLU or semilinear unit.

2. Boolean functions and their extension

We shall start by considering Boolean functions and establish a closed expression for their functionality which lends itself naturally to a formal, analogue extension. First, some notation: If X is a Boolean

variable then $\bar{x}$ will denote its value in the *polarised representation* with $\bar{x} \in \{-1,1\}$. Its value in the more usual binary representation will be denoted by $\underline{x}$, where $\underline{x} \in \{0,1\}$. (As a mnemonic for this notation note that the *un*polarised form is denoted with an *un*derscore.) The relationship between the two representations is given by the correspondence $0 \leftrightarrow -1$, $1 \leftrightarrow 1$, or

$$\bar{x} = 2\underline{x} - 1 \quad \text{and} \quad \underline{x} = \frac{1}{2}(\bar{x} + 1) \tag{1}$$

A Boolean function (defined with respect to the binary representation) is a mapping $f:\{0, 1\}^n \rightarrow \{0, 1\}$. This has the geometrical interpretation that each point in the domain gives the coordinates of a vertex or *site* on the n-dimensional hypercube so that the function is defined by a population of 0's and 1's at the sites of this cube. Sites will be denoted by lower case Greek letters μ, ν, ... and will be identified with the bit string or address $\underline{\mu}1,\underline{\mu}2,...\underline{\mu}n$ which gives its cube coordinates. The value at site μ will be denoted by S_μ.

The main feature of the geometric picture is that it illustrates the natural topology on the mapping domain, or address space, defined by the Hamming distance between addresses. Learning generalisation can then be though of as a due to clustering of similar site values on the hypercube. This is discussed more fully in Section (5.1).

2.1 A closed expression for Boolean Functionality

In order to give insight into the meaning of the general expression we first examine the case of a function with two inputs, $Y = F(X_1, X_2)$. Using the unpolarised form for both inputs and site values we have

$$\underline{y} = \underline{s}_{00}(1 - \underline{x}_1)(1 - \underline{x}_2) + \underline{s}_{01}(1 - \underline{x}_1)\underline{x}_2 + \underline{s}_{10}\underline{x}_1(1 - \underline{x}_2) + \underline{s}_{11}\underline{x}_1\underline{x}_2 \quad (2)$$

since the only one of these terms which survives for any input μ is just that which contains s_μ. Ullman (1973) discusses relations like (2) in the context of orthonormal expansion of Boolean functionality. However, by using the polarised representation in the input domain, we get a more symmetric form for this expression.

$$\underline{y} = \underline{s}_{00}\frac{(1 - \bar{x}_1)(1 - \bar{x}_2)}{2 \quad 2} + \underline{s}_{01}\frac{(1 - \bar{x}_1)(1 + \bar{x}_2)}{2 \quad 2} + \underline{s}_{10}\frac{(1 + \bar{x}_1)(1 - \bar{x}_2)}{2 \quad 2} + \underline{s}_{11}\frac{(1 + \bar{x}_1)(1 + \bar{x}_2)}{2 \quad 2} \quad (3)$$

In the general *n*-D case (3) becomes

$$\underline{y} = \frac{1}{2^n}\sum_{\mu}\underline{s}_{\mu}\prod_{i=1}^{n}(1 + \bar{\mu}_i\bar{x}_i) \quad (4)$$

2.2 Activation-Output Formulation

It is convenient with weighted node models (TLU, sigma-pi, etc.) to express the overall functionality in two parts. Thus, there is an activation $a = a(x_i, w_j)$ which is function of the inputs x_i and internal parameters or weights w_j, and the output $y = y(a)$ which is a function of the activation. Typically a is a linear weighted sum of the inputs and $y = \sigma(a)$ where σ is the sigmoid or logistic 'squashing' function. (4) may be cast into activation-output form by defining the activation in terms of the polaised site values $\bar{s}_\mu$ and using the relation (1) between the two representations.

$$a = \frac{1}{2^n}\sum_{\mu} \bar{s}_{\mu} \prod_{i=1}^{n} (1 + \bar{\mu}_i \bar{x}_i) \tag{5}$$

$$\underline{y} = \frac{1}{2}(a + 1) \tag{6}$$

Since a only takes on the values ± 1, the output relation (6) may be represented as a step function or the limiting sigmoid, formed by letting the gradient at zero tend to infinity as shown in Figure 1.

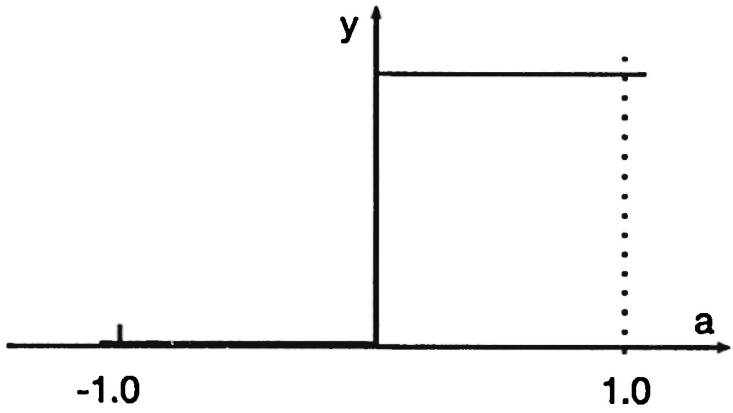

Fig. 1 Activation-output relation for Boolean functions using polarised binary representation for site values.

We now have the Boolean functionality in a form that lends itself to an analogue extension and have made the first point of contact with weighted node models.

2.3 Generalisation to analogue nodes

The motivation here is that networks of nodes with continuous site

values are amenable to analysis, since it makes sense to define derivatives of an output error with respect to the sites. Further, more training information may be captured by allowing more than the two site values of the Boolean case (see section 5.3 for fuller discussion).

We first define, quite formally, an analogue extension of (5) and (6), and then show how this maps onto a cube-based structure with stochastic dynamics. Let z_i and S_μ be two sets of real numbers with $z \in [-1, 1]$, and $S_\mu \in [-S_m, S_m]$. There is no interpretation at this stage of the μ as site addresses or the S_μ as site values. However, the index i is integral in the range 1,...,n and the μ are the set of bit strings $\mu_1, ..., \mu_n$. Define the real valued activation function $a(z_i, S_\mu)$ and the output $y(a)$ by

$$a = \frac{1}{S_m 2^n} \sum_{\mu} S_\mu \prod_{i=1}^{n} (1 + \bar{\mu}_i z_i) \tag{7}$$

$$y = \sigma(a) \tag{8}$$

y is now defined throughout the real-valued cube $[-1,1]^n$. A Boolean function may be recovered by restricting the z_i and the S_μ to the extremes of their intervals with $S_m = 1$.

Now it would be possible to construct a network of nodes defined by (7) directly but the processing involved in doing this would be considerable; for an n-input unit there are 2^n strings μ to be summed over. In the next section it will be shown how we may trade-off computational complexity with time in a stochastic version of this node type. We will call a node using the direct, analogue approach of (7) the *A-model*, and its stochastic counterpart, the *S-model*.

3. The stochastic models

3.1 Stochastic processing and unary representations

It is customary to represent numbers in a weighted positional scheme where the digit r_n at the nth position from the right stands for $b^{n-1}r_n$ where b is the representation base. In contrast, in the unary representation, each place in the number string has the same unit weight and may be filled with either one or zero. For example, the number 4 may be represented by any of the strings '1111', '1111000', '1011001'. One of the advantages of unary representations is their immunity to noise. Thus if we change one of the bits in a binary number then, depending on its position, we may alter its value by orders of magnitude. In a unary number with string length L, we alter the value by $1/L$ in any position.

Suppose now that a stream of bits is being generated with a fixed probability p of producing a '1' and we wish to find p by observing the resulting bit-stream. If there are N_1 '1's in the stream of length L, an estimate p^* of p is given by N_1/L. The representation of p is a stochastic unary one.

Since N_1 is binomially distributed with probability p, its standard deviation, std (N_1)given by

$$\text{std}(N_1) = \sqrt{Lp(1-p)} \tag{9}$$

so that

$$\text{std}(p^*) = \left[\frac{p(1-p)}{L}\right]^{\frac{1}{2}} \tag{10}$$

which varies as $1/\sqrt{L}$

The use of stochastic bit streams to represent numbers allows arithmetic to be done on pairs of streams with very simple hardware. Thus multiplication may be done by passing the streams through an AND gate, and addition performed (with some precaution) by ORing them. Mars and Poppelbaum (1981) have shown how to develop complete arithmetic units for conventional Von Neumann architectures using this kind of stochastic processing. Using this kind of number representation in an artificial neuron may therefore confer advantages in ease of hardware implementation.

In the following section we show how the A-model developed above may be instantiated with the help of stream processing and in section (5.1) how bit-streams may be used to promote generalisation.

3.2 The S-Model

Informally, the z_i of the A-model will be interpreted as defining the probability of a '1' appearing at the ith input to a cube which has, at its vertices, the site values S_μ. Estimates are made of a by time-windowing a stream of bits, each one determined stochastically by the site values visited at each time step. The node structure is shown in Figure 2.

Let $F:\{0,1\}^n \rightarrow [-S_m, S_m]$ be defined at the corners of the n-cube by associating each ordered string μ in the A-model with the cube site μ, so that $F(\mu) = S_\mu$. Next we suppose that, at each time step, a new *input address* is formed from the set of Boolean variables $\{X_i\}$. These are defined via a set of probability distributions determined by the $\{z_i\}$.

$$P_1(x_i) = \frac{1}{2}(1 + z_i) \quad \text{and} \quad P_0(x_i) = \frac{1}{2}(1 - z_i) \tag{11}$$

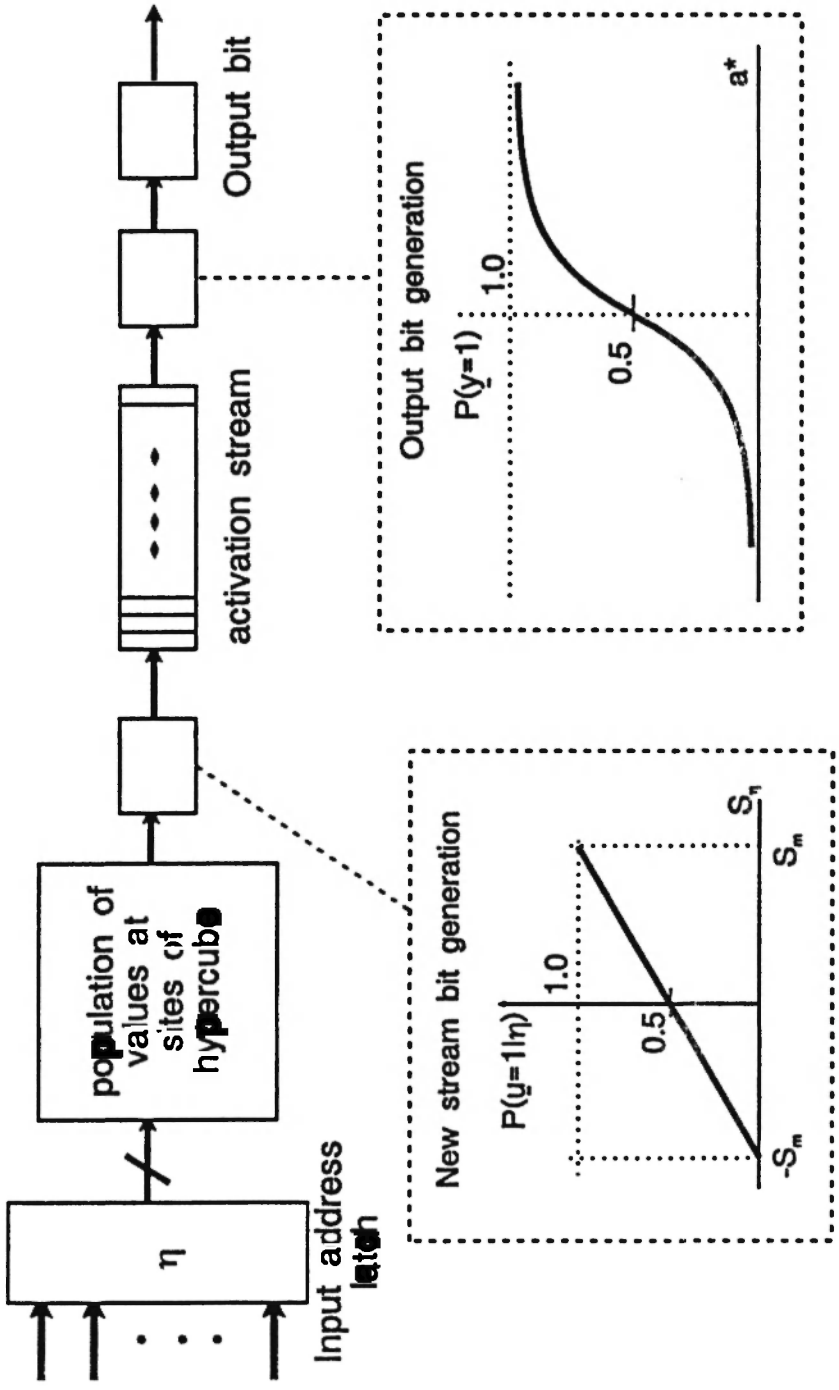

Fig. 2 S-model structure of Time Integration Node

where $P_{0,1}(x_i)$ is the probability that $x_i = 0,1$. Hence the probability $P_\mu(x_i)$ that the x_i is equal to the ith component of site μ is given by

$$P_{\mu_i}(x_i) = \frac{1}{2}(1 + \bar{\mu}_i z_i) \tag{12}$$

Now if no two inputs come from the same unit, the bit distributions on all inputs are independent. The probability $P(\mu)$ that the current input address locates site μ is then

$$P(\mu) = \frac{1}{2^n}\prod_{i=1}^{n}(1 + \bar{\mu}_i z_i) \tag{13}$$

(7) now becomes

$$a = \frac{1}{S_m}\sum_{\mu} S_\mu P(\mu) = \frac{\langle S_\mu \rangle}{S_m} \tag{14}$$

where $\langle\cdot\rangle$ denotes expectation value. Note that $-1<a<1$.

Let η be the current input address, then use S_η to generate a new *activation-stream* bit (see Fig. 2), according to

$$P(u = 1 \mid \eta) = \frac{1}{2}(S_\eta / S_m + 1) \tag{15}$$

Over many time steps, if the z_i are held constant

$$\langle u \rangle = P_1(u) = \frac{1}{2}(\langle S_\eta \rangle / S_m + 1) \tag{16}$$

so that using (14)

$$a = 2\langle u \rangle - 1 = \langle \bar{u} \rangle \tag{17}$$

If N_1 is the number of 1's in the activation stream which is L bits long, then an estimate for $<\underline{u}>$ *is* N_1/L. An estimate a^* for the activation is now given by

$$a^* = 2N_1/L - 1 \qquad (18)$$

We now use this to obtain an estimate $y^* = \sigma(a^*)$, of the output y. This in turn defined a distribution on a Boolean random variable Y, by $P_1(\underline{y}) = y^*$, which is used to communicate the output to the next layer. Assuming stationary conditions, the stream of output bits generated in this way will, over time, transmit an estimate of the value of y for this node.

The stream of bits appearing at the output may be likened to the trains of action-potentials transmitted along axons. Real neurones also time integrate their inputs (Kuffler, Nicholls & Martin, 1984) indicating that frequency information or probabilities are being used.

The nodes described in this section will be referred to as *Time Integration* nodes or TINs, in contrast with the *Direct Output* nodes or DONs discussed in the next section. Note that both types have *A*- and *S*- model descriptions. The DON is similar in function to the pRAM developed by Taylor and Gorse (Taylor, 1972; Gorse & Taylor, 1990) with a view to modelling a certain biological neural properties. However, the models and results in this paper, based on a PhD thesis (Gurney, 1989a), were obtained independently and have their origins in Boolean/PLN nets as described above.

3.3 An alternative node structure

Consider a feedforward net of m layers of TINs and suppose that the input probabilities at the first layer are changed. The new outputs at

the final layer will be estimated after mL time steps. During this transient period there is no simple causal relationship among the contents of the streams, for the most recent few bits in the stream of a unit are not just a result of the corresponding bits in previous layers, but also of the other bits in those layers that were generated at earlier times. If we are to take advantage of the correlation between short term fluctuations in the output of nodes in subsequent layers to provide training information (see section 5.3) then we require that there be *micro-temporal causality* throughout the net. This is the case with Reward Penalty training and a new algorithm based on system identification (Gurney, 1989a).

To this end we incorporate the non-linearity $\sigma(\cdot)$ into the stream bit generation and identify the output with the activation. Consider the *A*-model defined by

$$a = \frac{1}{2^n}\sum_{\mu} \sigma(S_\mu) \prod_{i=1}^{n}(1 + \bar{\mu}_i z_i) \tag{19}$$

so that in the *S*-model

$$a = \sum_{\mu} \sigma(S_\mu)P(\mu) = \langle \sigma(S_\mu) \rangle \tag{20}$$

The output y is just put equal to the activation a and

$$P(\underline{y} = 1 | \eta) = \sigma(S_\eta) \tag{21}$$

Thus, in the TIN, $y = \sigma(\langle S_\mu / S_m \rangle)$, whereas in the DON, $y = \langle \sigma(S_\mu) \rangle$. The S-model structure for the DON is shown in Fig. 3.

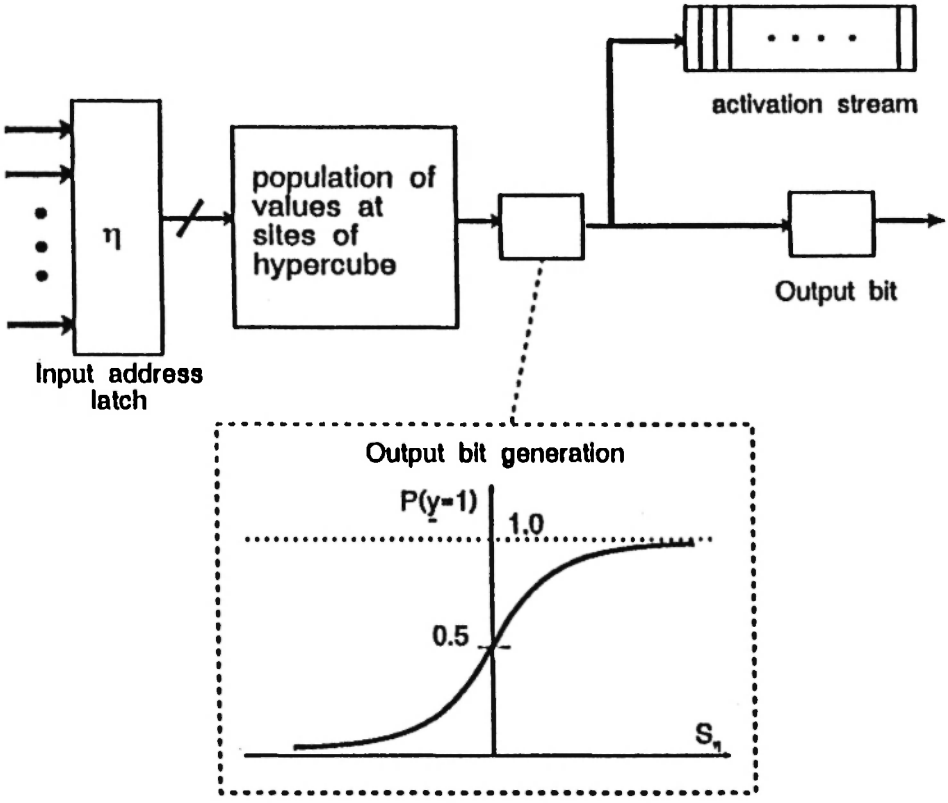

Fig. 3 S-model structure of Direct Output Node

In DONs, bits are produced and fed straight into the unit's fan-out without an intermediate stream, resulting in corresponding fragments of the a-stream being causally related. For the forward pass the a-stream is redundant but is required in training whenever the activation value is needed.

If the site values are quantised and there is no stochastic recording of activation estimates in a bit stream, the resulting node is the same as the Multi-level Probabilistic Logic Node (MPLN) developed by Myers and Aleksander (Myers & Aleksander, 1988).

3.4 Comparison between A- and S- Models

* The expressions for the activation are formally the same and, given the following correspondence, the two models are isomorphic.
  - In the A-model μ is just an ordered string of Boolean values, whereas in the S-model it is the address for a cube site.
  - The analogue input values z_j of the A-model are interpreted in the S-model as defining the probability of there being a '1' at the jth input.
  - In the A-model, Sμ are just real parameters used to define unit functionality; in the S-model, Sμ is associated with cube site μ.
* The A-model is deterministic - given a set of inputs then there is a unique output value. Equivalently, in the S-model - given the input statistics are stationary, the expectation values are well defined and remain constant. In this sense the S-model is 'deterministic'. If the statistics are not stationary, the bit-streams act as moving average filters.
* The A-model uses the interior of the n-cube which corresponds in the S-model to averaging over a set of samples from the cube corners.
* The A-model uses instantaneously available values whereas it may take many time steps for the S-model stream to fill and give a good activation estimate.
* The S-model lends itself to a physical implementation in RAM technology - see section 5.2.

4. Using more than one cube

One of the problems with the cube-based approach is that the storage requirement grows exponentially with the number of inputs; for example, a 32 input Boolean function requires 2^{32} bits. Intuitively it appears that the node functionality is far too rich here and that it could be pruned considerably while still maintaining a processing ability

useful in connectionist-type tasks. There is a simple way to do this which allows a similar formal description to that developed so far, has a biological analogue, and allows further contact to be made with weighted nodes.

Figure 4(a) shows a TIN or DON where the details of the particular activation-output (a-o) functionality have been suppressed; for TINs this will be an integrating stream followed by a sigmoid, for DONs there is no integration but an activation recording stream as in Figure 3. In 4(b) is shown a natural, linear extension of this where the total activation is the sum of site values from several cubes. 4(c) shows a further extension which includes simple, linear weight factors in the activation, thus allowing the use of these nodes in competitive nets. If there are m cubes each with n inputs, there are now mn inputs and $m2^n$ sites.

If the activation-output function is replaced by the identity and the cubes are boolean, then the structure in Figure 4(b) is the same as that of a single discriminator as used in the WISARD pattern recognition device (Aleksander, Thomas & Bowden, 1984).

The biological counterpart of the cube in these nodes is a localised cluster of synapses which are supposed to modulate each others activity in a multilineal way (Kandel & Schwartz, 1985). Each cluster is then considered sufficiently isolated that their resulting post-synaptic potentials may be summed. Further realism may be introduced by allowing delay elements between each cube and the final summation, representing the consequences of finite dendritic time constants.

5. Aspects of cubic nodes and their networks

5.1 Generalisation

In this section we shall restrict the discussion to the case of binary

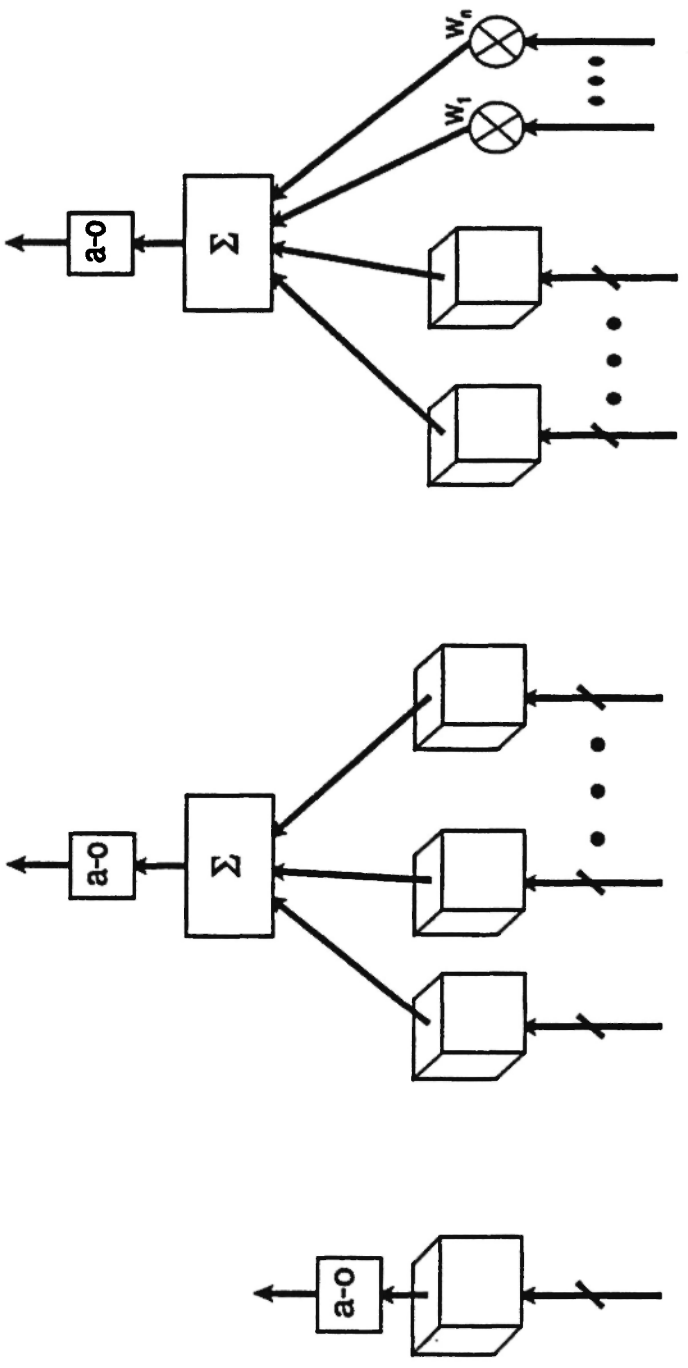

Fig. 4 Multiple Cube Nodes

inputs to clarify the situation, although all points made here are valid in the general case with suitable elaboration of detail.

Consider a cubic node which, in the untrained state, has all sites set to zero. Independent of the particular type of node, the output will be totally random with there being equal probability of a 1 or a 0. If this node is now trained on the two (Boolean) vectors, only the two sites addressed by these will have their values altered; any other vector will produce a random output and there has been no generalisation. This may be thought of as a price to pay for the increased functionality of this type of node. We shall call sites addressed by the training set *centre sites* or *centres*. In order to promote Hamming distance generalisation, sites close (in the Hamming distance metric) to the centres need to be trained to the same or similar values as the centres themselves. In terms of the cube structure there should be a clustering of site values around the centres. This may be done automatically during training or done offline by, for example, performing a Voronoi tessellation of the cube (Gurney, 1989a, 1989b). This latter type of process has recently been suggested independently by Aleksander and embodied in the GRAM (Aleksander, 1990). All online training methods rely on expanding the training set to include noisy copies of the original set, thereby visiting a selection of nearby sites. It is now shown how to use bit-stream structures (Fig. 5) at the input and output to automatically produce noise and promote generalisation.

At each input, new bits to the first hidden layer are obtained from the value in the stream by interpreting its contents as a unary representation of probability. Thus, if there are N_1 '1's in the stream the probability of the next input bit being a '1' is just N_1/L_{in} where L_{in} is the stream length. The contents of the ith stream are obtained by gradually shifting in the value $\underline{v}_i(t)$ the relevant bit position in the current external training vector $v(t)$ under control of a clock signal φ.

Suppose that $\underline{v}_i(t) \neq \underline{v}_i(t-1)$ and that the ith input stream is

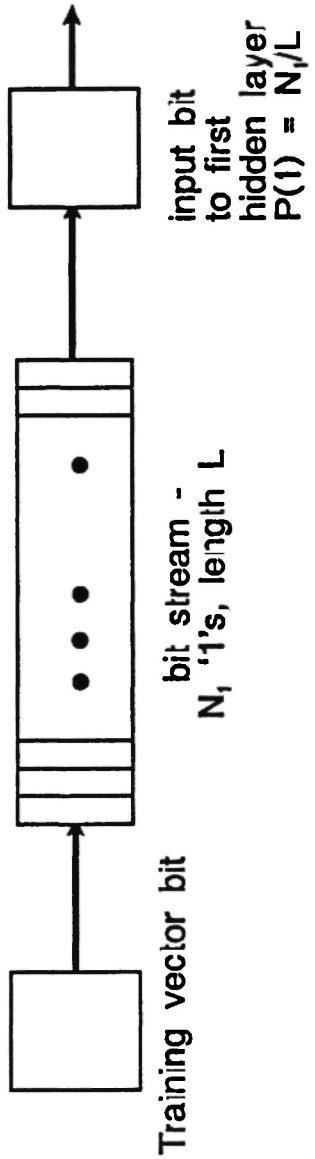

Fig. 5 Bit-stream structure for training set generalisation

saturated with the value $\underset{\sim}{v}_i(t-1)$, so that the input bits to the first hidden layer all take this value. Training is now occurring on the Boolean vector $\underset{\sim}{v}(t-1)$. At the next signal from φ the stream is shifted right and a new bit inserted with value $\underset{\sim}{v}_i(t)$. The distribution of new input bits now allows both '0' and '1' although still heavily weighted towards $\underset{\sim}{v}_i(t-1)$. Training now takes place with the new distributions before clocking φ again. Eventually the stream will be full with bit value $\underset{\sim}{v}_i(t)$ and training will take place with respect to the Boolean vector $\underset{\sim}{v}(t)$. At intermediate times, cube sites will be visited which are in the neighbourhoods of the centres defined by $\underset{\sim}{v}(t-1)$ and $\underset{\sim}{v}(t)$, and clustering will occur. Clearly, a similar stream mechanism will be required at the output layer in any supervised algorithm.

The stream-clustering technique described here fits naturally into the stochastic paradigm and may be thought of, alternatively, as extending the training set to include analogue vectors.

5.2 Implementation

The starting point here is the observation that the truth-table for any Boolean function may be realised using the locations in a RAM memory store. This was the basis for the WISARD pattern recognition device developed by Aleksander et al. (Aleksander, Thomas & Bowden, 1984). By using several RAMs tied to the same address latch we may implement a cubic node with more than two values at each site. This may be done by interpreting the memory word addressed as either a binary or a unary number. In both cases site values are restricted to the integers m in the range $-S_\mu \leq m \leq S_m$ and we have a *quantised S-model*. The effect of quantisation is to introduce noise into training, since we may not perform arbitrary changes in the site values as dictated by gradient descent analysis on the continuous models.

Clearly, the binary scheme is more memory efficient, requiring

$2^n \log 2 S_m$ as opposed to $2^{n+1} S_m$, but the unary approach is closer in spirit to a neural paradigm and has two advantages. Firstly, the disruption of any randomly chosen memory cell has an equally small effect whereas, in a binary representation, the destruction of the msb may erase a significant amount of training information. This points to the possible use of extremely high density RAM cell technology which is, by conventional standards, worthless because of its bit yield but which may be perfectly adequate for unary storage. Secondly, the summation required in MCUs may be performed simply by concatenating the individual cube outputs and interpreting the resulting string accordingly.

The unary approach will only be realistic if it can be shown that the number of site levels required in a quantised S-model is moderate. Work by the author (Gurney, 1989a) has demonstrated that as few as 10 levels may be adequate and this is supported by other work with cube-based nodes (Myers, 1989).

5.3 Training

It is not the intention here to discuss particular algorithms in detail but to provide references to some of those that have been reported and discuss some points about training cubic nodes in general.

It may be shown (Gurney, 1989a) that learning convergence can be proved for nets of cubic nodes under the Reward-Penalty (Barto & Jordan, 1987) and Backpropagation schemes. Myers and Aleksander (1988) have developed a Reward-Penalty style algorithm which, although having no explicit proof of convergence, has been shown empirically to work under a wide variety of data sets and network architectures. In a similar vein, R. Al-Alawi (1990) has developed a Backpropagation style algorithm. It is possible to show convergence under a hybrid algorithm which combines some of the features of the Myers-Aleksander scheme with the conventional Reward-Penalty

algorithm (Penny, Gurney & Stonham, 1990). Milligan (1988) discusses some strategies for training recurrent nets using ideas based on stimulated annealing. Filho, Bisset & Fairhurst (1990) have developed an algorithm which considers the search for unused sites as a goal for each node during training leading to their designation 'Goal Seeking Neurons' (GSNs).

Convergence for DONs may also be shown under a technique based on System Identification in control theory (Gurney, 1989a). Essentially, the short term fluctuations in the stream values of a DON provide 'test signals' for the network (the 'plant') which is parametrised by the derivatives of an error with respect to the site values. It is the requirement for causality or correlation between sub-streams that leads to the necessity for the Direct-Output-Node structure. It may be shown that special cases of this method reduce to Reward-Penalty training.

We now turn to some general properties of cubic nodes under training. As noted in section 5.1, training a cubic node with two vectors alone does not dichotomise the input space, or equivalently, there is no generalisation. On the other hand, setting the weights in a TLU always establishes a hyperplane across the n-cube resulting in a true classifier. This apparent disadvantage may be overcome using the techniques described in section 5.1 and may be viewed as the price to pay for an increased functionality, since the other sites are not automatically determined by two vectors.

There is however an advantage in training cube sites individually in that only one node parameter (weight or site value) is updated per training step. If node parameters are changed in parallel this means less hardware and, if done serially, less time used in a realisation of the node. For an MCU with m,n-input cubes, we have to update m parameters. This is still much better than the $m2^n$ weights in the equivalent sigma-pi unit.

In section 2.3, it was noted that multi-valued sites retain more training information. To see this, suppose that a site in a Boolean node (site values 0,1) has been trained with the sequence 1001001000001. Clearly we should register the preponderance of 0's by storing a 0 at the site addressed. However since the last digit in the sequence was a 1, this is what is retained, resulting in an incorrect description of training. If, however, we have a quantised site value with $S_m = 10$, then the final site value (assuming we initialise to 0) is -5, which has correctly stored the frequency of occurrence of 0's and 1's.

6. Comparison of weighted and cube-based nodes

We are now in a position to make points of contact between the 'classical' weighted-sum-of-inputs nodes and those based on cube structures.

6.1 Sigma-pi equivalence

By multiplying out the product terms in (7) we obtain the (A-model) activation for a TIN as series expansion.

$$a = \frac{1}{S_m 2^n}\{\sum_{\mu} S_\mu + \sum_{i=1}^{n}\sum_{\mu} S_\mu \bar{\mu}_i z_i + \sum_{i<j}\sum_{\mu} S_\mu \bar{\mu}_i \bar{\mu}_j z_i z_j + ,...,\} \tag{22}$$

which may be written as

$$a = \frac{1}{S_m 2^n}\sum_{k}\sum_{\mu} S_\mu \prod_{i \in I_k} \bar{\mu}_i z_i \tag{23}$$

where I_k is an index set drawn from the integers 1,...,n. There are 2^n such sets since each integer may be either included or not included in the set. Now put

$$c_\mu^k = \frac{1}{S_m 2^n} \prod_{i \in I_k} \bar{\mu}_i \tag{24}$$

so that

$$a = \sum_k \left\{ \sum_\mu S_\mu c_\mu^k \right\} \prod_{i \in I_k} z_i \tag{25}$$

This is now in sigma-pi form with weights w_k defined by

$$w_k = \sum_\mu S_\mu c_\mu^k \quad \text{for } 1 < k < N\text{: } N = 2^n \tag{26}$$

Thus given $\{w_k\}$, (26) is a set of N equations for the N site values $S\mu$. It is not difficult to show that the rows and columns of the determinant $|c^k\mu|$ are linearly independent implying that the latter is non-zero and that there is, therefore, a solution to these equations. This means that TINs are functionally equivalent to sigma-pi units. Although we have used the A-model for convenience, the equivalence extends to the S-model under the conditions of the A-S isomorphism discussed above.

A similar treatment may be given for DONs by using (19) as the starting point. The resulting expressions may be obtained from those above by replacing S_μ by $\sigma(S_\mu)$ and omitting the factor S_m.

6.2 MCUs, TLUs and perceptrons

Although a semilinear unit is subsumed functionally by a single cubic node, a more direct analogy may be drawn with a special case of the MCU. Consider an MCU with m TIN type 1-cubes so that each 'cube' has only 2 sites. Denote quantities in the ith cube by an i superscript. Then, working with the A-model

$$a^i = \frac{1}{2S_m}[S_0^i(1 - z_i) + S_1^i(1 + z^i)] \tag{27}$$

Now put

$$\frac{-\theta}{m} = S_1^i + S_0^i \tag{28}$$

$$w_i = S_1^i - S_0^i \tag{29}$$

By a suitable choice of S_m we may solve this pair of equations for any θ and w_i. The activation of the ith cube is then

$$a^i = \frac{-\theta}{2mS_m} + \frac{w_i z^i}{2S_m} \tag{30}$$

and that of the whole node, a is given by

$$a = \sum_{i=1}^{m} a^i = \frac{1}{2S_m}\sum_i w_i z^i - \frac{\theta}{2S_m} \tag{31}$$

We may rescale the activation by $2S_m$ by introducing a parameter in the sigmoid exponent of $1/2S_m$. The scaled node activation a' is then just

$$a' = \sum_{i=1}^{m} w_i z^i - \theta \qquad (32)$$

which is just that for a TLU. Thus the latter may be thought of as a special case of an MCU with TIN cubes.

Figure 6 shows a perceptron (Rosenblatt, 1962). The φ units at the input are general Boolean functions. These are fixed and it is the weights w_i which are varied during training. Formally the structure looks similar to an MCU and may be obtained from it by fixing the weights at unity, restricting the (now trainable) site values to ±1, and using a DON type output (sigmoid) which has an infinite slope at the threshold.

In summary, the routes between the various node type are illustrated in Figure 7.

7. Summary

By a natural extension of the expression for Boolean functionality to continuous site values and inputs, we obtain a node which has a time stochastic realisation and which has a tractable mathematical analysis in terms of proof of supervised learning convergence. Generalisation may be promoted using a structure (the bit stream) which is already an integral part of the node's stochastic apparatus and may be though of as a process of site clustering on the hypercube. The use of several cubes overcomes the problem of exponentially growing resources with increased fan-in, and provides a natural contact with TLUs and semilinear units.

Cubic nodes offer an alternative approach to obtaining sigma-pi functionality. The use of hypercube site values, rather than the weights of the sigma-pi expression, offers the advantage of only having to train one parameter at each step and an immediate implementation in digital RAM hardware. The relation between site values and weights is given

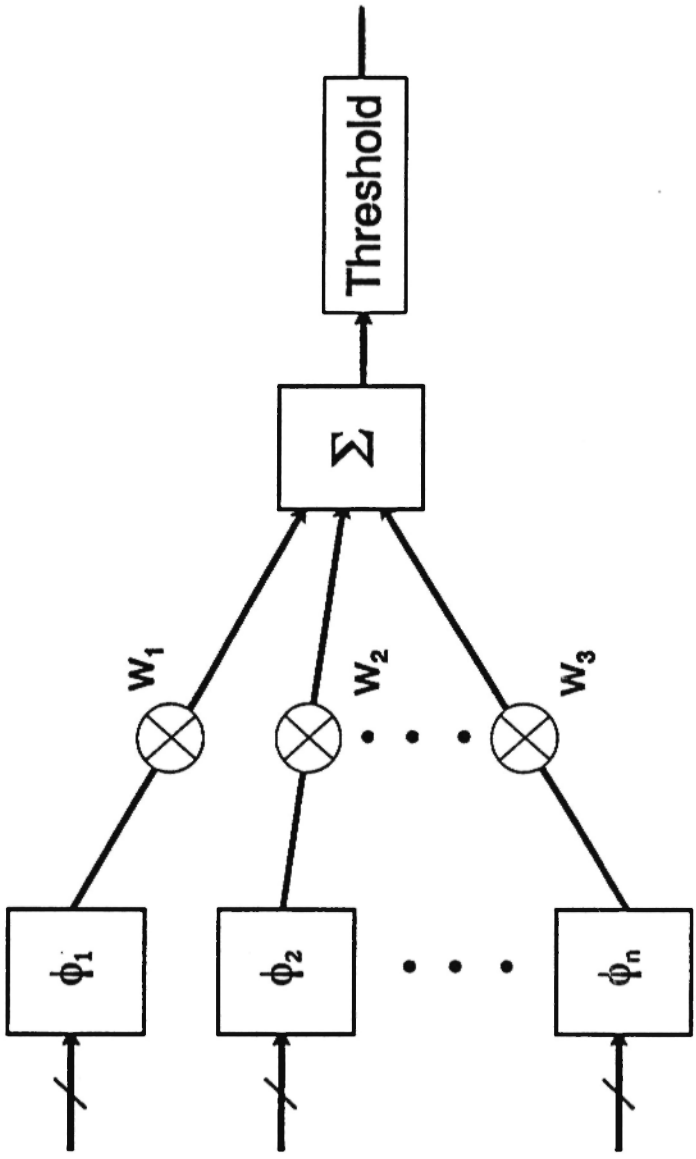

Fig. 6 Perceptron

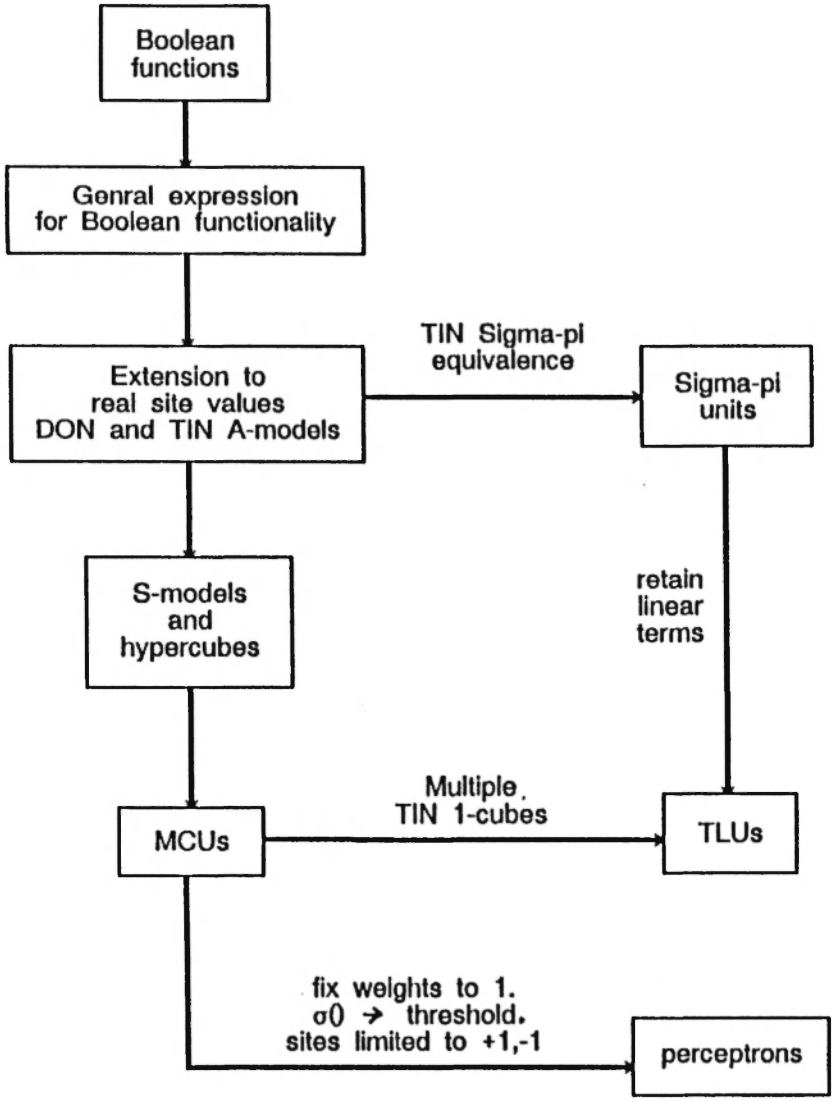

Fig. 7 Relationship between node types

by (26). MCUs limit the order of the multilinear input terms in a biologically plausible way.

Table 1 summarises some of the comparisons made for TLUs (or semilinear units), Sigma-pi units and cubic nodes.

	TLUs	SIGMA-PI	CUBIC NODES
functionality	Functionality limited to linear separability	General Boolean. Multilinear analogue	General Boolean. Multilinear analogue
Implementation	Convolve inputs and weights (implies multiplication)	Convolve inputs and weights (implies multiplication)	RAM access and sum
Generalisation	Automatic	Automatic	achieved on-line via streams or off-line via cube tesselation
Training	Train n weights at each step	Train 2^n weights at each step	Train single site at each step

Table 1: Comparison of node types.

The sigma-pi viewpoint of multi-valued site RAM-based nodes places them in the context of mainstream connectionism and may be one way to foster their acceptance by a lager part of the neural-net community.

References

Al-Allawi, R. and Stonham, T.J. 1990 "A Training Strategy and Functionality Analysis of Multi-Layer Boolean Neural Networks", Preprint available from Dept. Elec. Eng. Brunel University, Uxbridge Middx.

Aleksander, I. 1990 pp 2-5 in "Parallel Processing in Neural Systems and Computers", Eds. Eckmiller et al., North Holland.

Aleksander, I., Thomas, W.V. and Bowden, P.A. 1984 "WISARD a radical step forward in image recognition," *Sensor Review*, pp. 120 124, July 1984.

Aleksander, I. and Stonham, T.J. 1979 "Guide to pattern recognition using random-access memories," *Computers and Digital Techniques*, Vol. 2, pp. 29-40.

Aleksander, I. 1973 "Random Logic Nets: Stability and Adaptation," *International Journal Man-Machine Studies*, Vol. 5, pp. 115-131.

Austin, J. 1987 "ADAM: A Distributed Associative Memory For Scene Analysis," IEEE International Conference on Neural Nets, San Diego.

Barto, A.G. and Jordan, M.I. 1987 "Gradient Following Without Backpropagation in Layered Networks", Proceedings of 1st IEEE International Conference on Neural Networks, Vol. 2, San Diego.

Filho, E.C.D.B.C. and Fairhurst, M.C. 1990 "A Goal Seeking Neuron for Boolean Neural Networks" ICCN-90, Paris

Gorse, D. and Taylor, J. 1990 "A General Model of Stochastic Neural Processing," *Biol. Cybern.*, Vol. 63, pp. 299-306.

Gurney, K.N. 1989a "Learning in Nets of Structured Hypercubes," PhD Thesis; available as Technical Memo CN/R/144 in the Dept. of Electrical Engineering, Brunel University.

Gurney, K.N. 1989b "Training Nets of Boolean Functions with Hidden Units Using Simulated Annealing' in "Neural Networks from Models to Applications" Edited by L. Personnaz and G. Dreyfus, IDSET, Paris.

Kandel, E.R. and Schwartz, J.H. (eds), 1985 "Principles of Neuroscience", 2nd edition, Elsevier.

Kuffler, S.W., Nicholls, J.G. and Martin, A.R. 1984 From Neuron to Brain, Sinauer Associates Inc., Sunderland, Massachusetts.

Mars, P. and Poppelbaum, W.J. 1981 "Stochastic and Deterministic Averaging Processors," Peter Peregrinus on behalf of IEE.

Martland, D. 1987 "Behaviour of Autonomous, (Synchronous) Boolean Networks," Proceedings of IEEE 1st International Conference on Neural Networks, San Diego, Vol. 2.

Milligan, D.K. 1988 "Annealing in RAM-based Learning Networks," Brunel University Technical memorandum CN/R/142.

Myers, C.E. and Aleksander, I. 1988 "Learning Algorithms for Probabilistic Neural nets," 1st INNS Annual Meeting, Boston.

Myers, C.E. 1989 "Output Functions for Probabilistic Logic Nodes," First IEE International Conference on Neural Networks.

Penny, W.D., Gurney, K.N. and Stonoham, T.J. 1990 "Reward-Penalty Training for Logical Neural Networks," IASTED International Conference on AI Applications and Neural Networks, Zurich.

Rosenblatt, F. 1962 "Principles of Neurodynamics," Spartan Books.

Rumelhart, D.E. and McClelland, J.L. (Eds.) 1986 Parallel Distributed Processing, MIT Press.

Taylor, J.G. 1972 "Spontaneous Behaviour in Neural Networks," Journal of Theoretical Biology, Vol. 36, pp. 513-528.