Pergamon

Neural Networks, Vol. 9, No. 4, pp. 603–625, 1996

Printed in Great Britain
0893–6080/96 $15.00+.00

PII:S0893-6080(96)00015-9

CONTRIBUTED ARTICLE

Adaptive Critic for Sigma–Pi Networks

RICHARD STUART NEVILLE AND THOMAS JOHN STONHAM

Brunel University

(*Received* 18 *July* 1994; *accepted* 4 *December* 1995)

Abstract—*This article presents an investigation which studied how training of sigma–pi networks with the associative reward-penalty (A_{R-P}) regime may be enhanced by using two networks in parallel. The technique uses what has been termed an unsupervised "adaptive critic element" (ACE) to give critical advice to the supervised sigma–pi network. We utilise the conventions that the sigma–pi neuron model uses (i.e., quantisation of variables) to obtain an implementation we term the "quantised adaptive critic", which is hardware realisable. The associative reward–penalty training regime either rewards, $r = 1$, the neural network by incrementing the weights of the net by a delta term times a learning rate, α, or penalises, $r = 0$, the neural network by decrementing the weights by an inverse delta term times the product of the learning rate and a penalty coefficient, $\alpha \times \lambda_{rp}$. Our initial research, utilising a "bounded" reward signal, $r^* \in \{0, \ldots, 1\}$, found that the critic provides advisory information to the sigma–pi net which augments its training efficiency. This led us to develop an extension to the adaptive critic and associative reward–penalty methodologies, utilising an "unbounded" reward signal, $r^* \in \{-1, \ldots, 2\}$, which permits penalisation of a net even when the penalty coefficient, λ_{rp}, is set to zero, $\lambda_{rp} = 0$. One should note that with the standard associative reward–penalty methodology the net is normally only penalised if the penalty coefficient is non-zero (i.e., $0 < \lambda_{rp} \leqslant 1$). One of the enigmas of associative reward–penalty (A_{R-P}) training is that it broadcasts sparse information, in the form of an instantaneous binary reward signal, that is only dependent on the present output error. Here we put forward ACE and A_{R-P} methodologies for sigma–pi nets, which are based on tracing the frequency of "stimuli" occurrence, and then using this to derive a prediction of the reinforcement. The predictions are then used to derive a reinforcement signal which uses temporal information. Hence one may use more precise information to enable more efficient training. Copyright ©1996 Elsevier Science Ltd*

Keywords—Sigma–pi, Adaptive critic, Associative reward-penalty, Multi-cube, Reinforcement, Dynamic programming.

1. INTRODUCTION

Biological neurons and synapses have information processing capabilities that make use of both short- and long-term information. We build on this fact by utilising longer term information extracted from the model of reality (F). The model of reality may be:

(1) a simulation model of the environment or reality (R);
(2) a network trained to simulate the environment; or
(3) a set of stimuli and actions, which approximate the environment's behaviour.

This information is used in order to approximate dynamic programming (Werbos, 1992a), where dynamic programming (DP) relates to the optimisation of a utility function (u) in a noisy and non-linear environment. The method used to implement the approximation of DP is called the adaptive critic methodology, where a critic is placed hierarchically above an action network which is being trained. An action network is a net that inputs current state information (x_i or F or R) and outputs the action vector $u_{(t)}$.

The adaptive critic's task is to produce another output function J, given input stimuli from the external model F and a utility function U (Figure 1), where J is termed the secondary or strategic utility function. Dynamic programming (DP) is used to maximise the function U over time by maximising the J function in the immediate future (Werbos, 1989). The adaptive critic is required to approximate DP by using F as an input to optimise J over a short time period and hence U over a long time period. The critic implements this by recording short-term

Requests for reprints should be addressed to Richard Stuart Neville, Dept. of Electrical Engineering, Brunel University, Uxbridge, Middlesex UB8 3PH, UK.

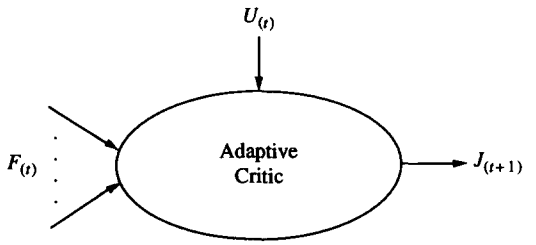

FIGURE 1. Diagrammatic representation of functional inputs and outputs of an adaptive critic

fluctuations of F and using these and the utility factory U to maintain longer-term metrics (J) in order to advise an action network.

The adaptive critic element is normally used in a control environment that requires temporal information in order to control a dynamic system. It may also be used for the more usual pattern recognition tasks, which we shall cover in the following paragraphs. Utilising the critic for these tasks still requires the optimisation of a longer term utility function U and the evaluation of a short-term strategic utility function J, where the utility function U may be a global metric used to advise an action net performing a credit assignment task (Minsky, 1961).

The format of the article is as follows; initially we introduce our area of research. Then we state our Rationale. The associative reward–penalty training regime is then introduced. Then we review the adaptive critic for semi-linear units. An introduction to logical neural networks is then given. The sigma–pi model is then presented. This is followed by a description of associative reward–penalty for sigma–pi networks. We then present a quantised adaptive critic. This is followed by a methodology for dealing with the quantised adaptive critic's enigma, which is the exponential growth of resources as the number of inputs to the critic increases. In the final simulation work we contrast the adaptive critic method with the more conventional associative reward–penalty paradigm. In this final work we also show that a version of the adaptive critic which utilises an "unbounded" internal reinforcement signal promotes optimal learning efficiency.

2. RATIONALE

The training of sigma–pi networks (Gurney, 1989) utilising associative reward-penalty (A_{R-P}) (which is presented in Section 3) can be a time consuming process. One of the contributory factors for this may be the very limited band-width of information the global scalar reinforcement, $r \in \{0, 1\}$, signal provides when it is broadcast to the action network. The scalar value does not advise the action network by utilising secondary information such as past data obtained from the environment R, which may aid the net in its learning task. The (A_{R-P}) algorithm has no means of facilitating the use of cause-and-effect information (Werbos, 1989) to make it possible to give credit to good actions more precisely than one could using an error driven reinforcement scheme alone (Werbos, 1989). We investigate the quantised ACE to enable us to increase the associative reward–penalty learning rule's efficiency and to enable us to re-formulate the ACE into a digital methodology so that it is hardware realisable. The following paragraphs relate how this may be done utilising what is known as an adaptive critic as an adviser for an action network (Neville, 1993; Neville & Stonham, 1993).

3. ASSOCIATIVE REWARD–PENALTY TRAINING

3.1. Historical Overview of Associative Reward–Penalty

The term "reinforcement" comes from experiments on animal learning in psychology. Reinforcement refers to increasing the probability of the occurrence of the correct response to a specific event. Barto (1992) states that the basic premise has root in the classical "law of effect" of Thorndike (1911). The associative reward–penalty (A_{R-P}) learning rule was derived from research by Barto in 1987, his basic work entailed using associative reward–penalty applied to a task, e.g., where nonlinear associative mapping are to be learnt by a feedforward network. Barto and Jordans' work of 1987 on associative reward–penalty (A_{R-P}) followed research by Williams (1986, 1987a, b) on reinforcement training methodologies.

The initial research on RAM-based digital sigma–pi networks utilising the associative reward–penalty paradigm was carried out by Gurney (1989) and was extended by Neville and Stonham (1994a, b, c, 1995) and Neville and Stonham (1995a, b, c). Complementary work on similar models (i.e., pRAMs) was done by Gorse and Taylor (1990a, b, 1991), who also utilise the reward–penalty methodology.

A unification of the sigma–pi model and the reward–penalty algorithm has been presented formally by Gurney (1992a, b, 1993). Investigations by Neville (1990), into the use of reward–penalty for on-board training of VLSI hardware implementation of sigma–pi networks, initiated the work of Hui et al. (1992a, b) into cascadable sigma–pi nets. Complementary work in this area has been carried out by Clarkson et al. (1992), using the pRAM units of Gorse and Taylor (1990a).

3.2. The Basis of Associate Reward–Penalty

The associative reward–penalty (A_{R-P}) algorithm utilises a binary (scalar) reward signal "r" which is globally broadcast across a network. The reinforcement signal "r" is then utilised by each unit in the net to determine their weight updates. The premise is that the stochastic nodes in the net are given credit, or reinforcement, if the net gives a "successful" output. The net is given a debit or penalised if its output was wrong (Minsky & Papert, 1969).

The associative reward–penalty training algorithm has been used in a supervised manner on feedforward networks (Barto & Jordan, 1987). The networks were multi-layered, with input units which had clamped values specified by sources external to the network. The input units feed what were termed "hidden" units, i.e., they were not available to the outside world. The hidden units communicate with other hidden units in the net hierarchically above them. The final layer of hidden units, as there may be more than one, feed the output units. The output units are available to the outside world and are also known as "visible" units.

Figure 2 shows a network of stochastic units in its training environment and the communication between the network and its environment. The operation of the network and the environment is described as a set of steps:

Step 1. The environment randomly selects an input pattern for the network from a set of patterns.

Step 2. Once the input has been selected an output action (pattern) associated with this CLASS of input pattern is also selected. The input pattern is presented to the net and, on a layer by layer basis, the activation passes through the network from the input to the output.

Step 3. When all the units at the output of the net have selected their action, a reward signal "r" is calculated.

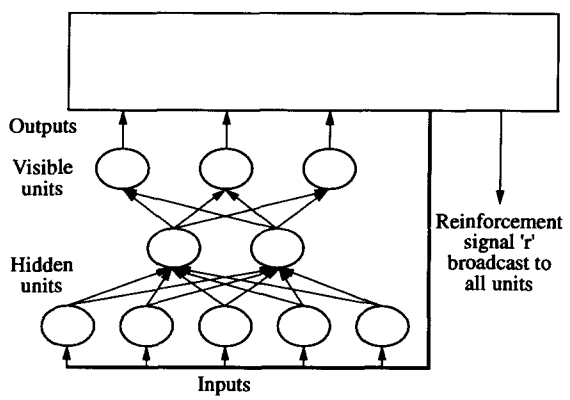

FIGURE 2. Stochastic network and training environment.

Step 4. Each unit changes its internal state according to some specified function of its current state, the action just chosen, its input, and the reward signal.

3.3. Associative Reward–Penalty for Stochastic Semi-linear Nodes

The intention of this section is to introduce the original concept of associative reward–penalty learning rules for stochastic semi-linear nodes. The input units, k, distribute the input stimuli to the hidden units which have real-valued inputs, x_k, each of which is associated with a weight, w_{jk}, in a multiplicative manner. The resultant products from all the inputs are summed, to obtain the activation a_j, and passed through a semi-linear function such as the sigmoid or squash function—this produces an output, y_j. The hidden units of the net are stochastic in nature as the binary output, y_j, is defined from the probability of firing [which is identical to that of the Boltzmann units, Hinton et al. (1984)]. Hence the hidden units' output are defined by

$$P(y_j = 1) = \sigma(a_j) \text{ or } P(y_j = 0) = 1 - \sigma(a_j). \quad (1)$$

The output units are deterministic semi-linear units identical to those of Rumelhart et al. (1986), whose output is given by:

$$y_i = \sigma(a_i) = \frac{1}{1 + e^{-a_i/\rho}} \quad (2)$$

where

$$a_i = \sum_j w_{ij} y_i. \quad (3)$$

Barto et al. (1987) trains the output units in the manner of Rumelhart et al. (1986), who uses a "delta", δ, rule. The delta rule adapts each output node, i, given w_{ij} is its weight, in the following manner

$$\Delta w_{ij} = \alpha(y_i^* - y_i)\sigma'(a_i)y_j \quad (4)$$

where y_i is the response of the deterministic output unit i to the input pattern; $y_i^* \in [0, 1]$ is the desired response of unit i supplied by the teacher; $\sigma'(a_i) = \sigma(a_i)(1 - \sigma(a_i))$ is the derivative of the sigmoid function σ evaluated at a_i, and α is the learning rate constant determining the step size.

One should note that, unlike the back-propagation error method, the error for a given input pattern is a random variable, due to the stochastic nature of the hidden units.

In order to train the hidden units an error metric is utilised to define the reinforcement signal "$r \in \{0,1\}$"; this is normally the inverse of the mean-square error, e_0 [i.e., of the form given in Section 6.2 eqn (34)] of the output units.

$$r = \begin{cases} 1 & \textit{with probability} \quad 1 - e_0 \\ 0 & \textit{with probability} \quad e_0 \end{cases}. \tag{5}$$

The reward signal is then used to assign incremental or decremental changes to the weights.

$$\Delta w_{jk} = \begin{cases} \alpha[Y_i - \sigma(a_j)]x_k & \text{if } r = 1 \\ \alpha\lambda_{rp}[1 - Y_i - \sigma(a_j)]x_k & \text{if } r = 0 \end{cases} \tag{6}$$

where $\alpha > 0$ is the learning rate, which defines by how much the weight is incremented, whilst $0 \leqslant \lambda_{rp} \leqslant 1$ is a penalty coefficient that defines the amount the weight is decremented, x_k is the real-valued input and a_j is the real-valued activation of the hidden unit. This leads to a non-symmetrical learning regime. It has been reported by Barto et al. (1987) that the lambda, λ_{rp}, term enables the algorithm to avoid "local minima". Gurney (1989) suggests that non-zero lambda introduces noise into the learning process and helps avoid absorption into "local minima".

The stochastic nature of these nets, which leads to non-deterministic searching of the state space, is derived from the probabilistic interpretation of the activations of the hidden units $P(Y_j = 1) = \sigma(a_j)$. Hence when the net's stochastic response is correct, i.e., when $r = 1$, then the weights are incremented in order to promote the probability of outputting the same response associated with a given stimulus.

4. ADAPTIVE CRITIC

The adaptive critic (Barto et al., 1983; Barto, 1992; Werbos, 1992a) in Figure 3 is placed hierarchically above another network, which it advises. The critic exploits cause-and-effect information; this enables credit to be given to good actions $(u_{(t)})$, more precisely than standard associative reward–penalty (A_{R-P}) error driven learning (Werbos, 1989). The adaptive critic element (ACE) supervises itself by detecting changes in the environment $(R_{(t)})$ (Myers, 1990). The guidance the ACE provides takes the form of a prediction (Barto et al., 1983) or utility function (Werbos, 1992a) which the ACE maximises over time.

The critic may be used to optimise either a utility function, a performance index or a cost function (Werbos, 1992a). In our case, the critic optimises a reinforcement signal which is used to advise an action network of changes in the value of a performance criterion. This type of predictive system is commonly used in control theory (Barto et al., 1983).

In Barto et al.'s original research (1983), the critic or functional network outputs an estimate of J, which is used in the reinforcement of lower-level networks. Their original functional network, in what has been termed their two-net problem (Werbos, 1989), estimates J and is called a "critic", because its main function is to criticise or evaluate the results produced by the action network, in order to permit adaptation of the action network. Barto's method may be thought of as a temporal difference (TD) method (Werbos, 1992c) as Barto utilises data that relate to past and present events to enable a payoff metric to be optimised, where the payoff was used as a "prediction" or "expectation" of a future reinforcement (Myers, 1990). The prediction values are calculated with reference to the ACE's input eligibility traces, where the eligibility is a trace of events over time (Barto et al., 1983).

The eligibility trace, Figure 4, may be described as follows; given a pathway between two neurons, the pathway reaches maximum eligibility a short time after the occurrence of a non-zero input signal on that pathway. The eligibility metric has been used by Barto et al. (1983) to update the weights of their action network, where the eligibility of a pathway relates to what extent the weight on that input pathway should be modified.

The input eligibility traces, $(\bar{x})$, are averages, where the bar $(\bar{\cdot})$ denotes an exponential average over time. The relationship relates the future eligibility trace, $\bar{x}_{(t+1)}$, to the present eligibility trace, $\bar{x}_{(t)}$, and the present input, $x_{i(t)}$, in an

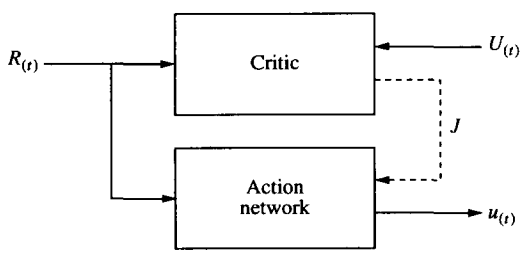

FIGURE 3. Adaptive critic advising an action network.

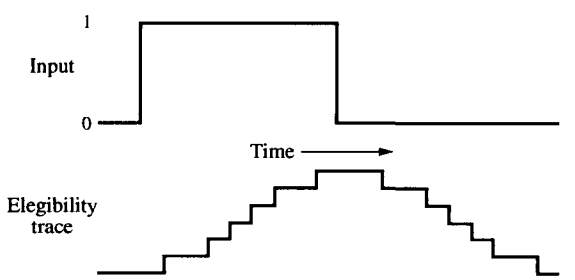

FIGURE 4. Diagrammatic representation of input eligibility trace.

exponential relationship defined by eqn (9), which is a recurrence relationship. The adaptive critic we utilise predicts an internal reinforcement signal based on the present external reinforcement signal and its past and present prediction values. Each input to the adaptive critic element is given its own trace. These input eligibility traces increase when the input is active and decrease to zero with time in the absence of future activity, hence the most recently increased eligibility traces would affect the production of an expectation most strongly (Myers, 1990).

The algorithmic sequence the adaptive critic follows is: given an external reward signal at time T, the critic then deduces an internal reinforcement based on the external reinforcement, the present and past predictions. The future prediction value is then derived as a function of the input eligibility trace. Finally all the input eligibility traces are updated.

4.1. Complementary Work

The initial research of Barto et al. (1983) on "neuronlike adaptive elements" utilised an associative search element (ASE) as the action network. The ASE was advised by an adaptive critic element (ACE). The ACE receives the externally supplied reinforcement signal which it uses to derive an improved reinforcement signal which it sends to the ASE. The central idea behind the ACE algorithm is that predictions are formed that predict not just reinforcement but also future predictions of reinforcement.

The ACE, Figure 5, has a reinforcement input pathway, n pathways for non-reinforcement input, and a single output pathway. The external reinforcement, $r_{(t)}$, denotes the real-valued reinforcement at time t, and let $x_{i(t)}$, $1 \leqslant i \leqslant n$, denote the real-valued signal on the ith non-reinforcement input pathway at time t. Each non-reinforcement input pathway, i, has a weight with real-value $v_{i(t)}$ at time t. The internal reinforcement output, $\hat{r}_{(t)}$, is the improved reinforcement signal that is used by the ASE in place of r. The ACE determines $\hat{r}_{(t)}$ by evaluating a prediction, $P_{(t)}$, of eventual reinforcement that is a function of the input vector, $X_{(t)}$.

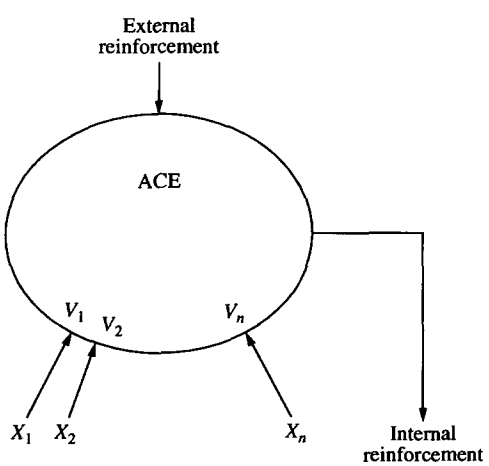

FIGURE 5. Adaptive critic element used by Barto.

The prediction is given by:

$$P_{(t)} = \sum_{i=1}^{n} v_{i(t)} x_{i(t)}. \tag{7}$$

Then the weights, v_i, are updated to enable $P_{(t)}$ to converge to an accurate prediction. The updating rule Barto used for the weights of the ACE was:

$$v_{i(t+1)} = v_{i(t)} + \beta[r_{(t)} + \gamma P_{(t)} - P_{(t-1)}]\bar{x}_{i(t)} \tag{8}$$

where $0 \leqslant \beta \leqslant 1$ is a positive constant determining the rate of change of v_i and $0.0 < \gamma \leqslant 1.0$, is a constant that has been called a "discount factor" by Witten (1977), which enables the eventual extinction of predictions in the absence of external reinforcement. The prediction is self-sustaining if $\gamma = 1.0$, but if it is less than 1.0 the prediction decays in the absence of external reinforcement, Barto et al. (1983) used a discount factor of $\gamma = 0.95$ for his research.

The trace, $\bar{x}_i$, acts as a type of eligibility trace. The input pathway gains positive eligibility whenever a non-zero signal is present on that pathway. The input eligibility trace was computed using the following linear difference equation:

$$\bar{x}_{i(t+1)} = \lambda_{\text{ace}} \bar{x}_{i(t)} + (1 - \lambda_{\text{ace}}) x_{i(t)} \tag{9}$$

where $\lambda_{\text{ace}}, 0 \leqslant \lambda_{\text{ace}} \leqslant 1$ determines the trace decay rate and $x_{i(t)}$ is the present input. The ACE's output, the improved or internal reinforcement signal, was computed from the past and present predictions as follows:

$$\hat{r}_{(t)} = r_{(t)} + \gamma P_{(t)} - P_{(t-1)}. \tag{10}$$

The $\hat{r}$ was substituted for r, the original external reinforcement, when the weights of the action network were updated. The pole-balancing problem, which Barto (1983) uses to demonstrate the ACE, utilised an r of zero throughout the training steps and it became -1 when failure occurred, i.e., the pole would not balance.

Werbos (1989, 1990, 1992a, b, c) re-evaluates the premise of the adaptive critic, to relate the methodology to a wider area of research.

Werbos observes that the adaptive critic method approximates dynamic programming, Figure 6, i.e., it is used to maximise a utility function in a noisy, non-linear environment. He denotes the critic's

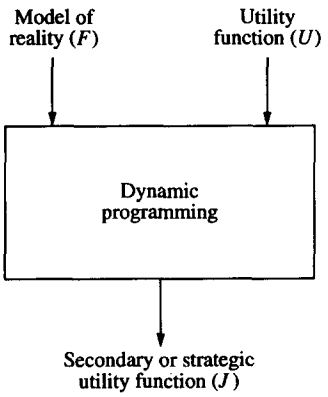

FIGURE 6. Diagrammatic representation of dynamic programming.

inputs, $R_{(t)}$ (or F), as the current description of reality (x_i input vector) and the utility function $U_{(t)}$ (which is equivalent to the reinforcement r). The critic outputs an estimate of the future utility, J, across $(t+1)$ future time, where J is a payoff metric which is used with the adaptive critic (J is optimised over time) and where U is maximised over time by maximising J in the immediate future. Werbos' research in this field (1989, 1990, 1992a, b, c) has extended the basic premise of the adaptive critic and he has presented several variations on the adaptive critic theme which use error back-propagation as the means of evaluating the utility function.

5. INTRODUCTION TO LOGICAL NEURAL NETWORKS

The majority of researchers into artificial neural networks utilise neuron models that implement a linear sum of the weights times their input stimuli. This sum is then passed through an activation–output function which is normally a sigmoidal transfer function. These units are termed "semi-linear" as the shape or "linearity" of the transfer function is defined by ρ [re. Section 6 eqn (25)] which is a positive parameter that defines the shape of the curve. This may be set to a hard-limiter if $\rho \rightarrow 0$ or with $\rho \rightarrow 0.4$ the curve becomes "semi-linear", where both the input and output of these units are real-valued.

A different perspective has been taken by some researchers (e.g., Aleksander, 1989a, b, 1990; Gorse & Taylor, 1990a, b, 1991; Gurney, 1992a, b, 1993; Neville & Stonham, 1994a, 1995; Neville et al., 1995a, b, c) who use what may be specified as "digital networks", which have the ability to implement the node functionality in hardware using random access memories (RAMs). The main driving force behind their research is derived from the fact that the mainstay of computational devices used today are "digital" in nature. These digital computational devices process analogue information by first digitizing the input (e.g., coding the analogue signal using an analogue to digital (A/D) converter) then processing the signal and finally converting the digital signal back to analogue (e.g., decoding the digital signal using a digital to analogue (A/D) converter). The whole ethos of these RAM based units is that they enable Boolean functions to be implemented as look-up tables in a RAM. Hence logic nodes' inputs and outputs are binary.

The basic difference between logical nodes and semi-linear nodes, which sum the product of the inputs and their weights, is that logical nodes respond to their input patterns in addressable locations; the locations contain either a logical "1" or "0", which is set/reset during a training phase. The output is then defined as the binary value stored at the location.

The following paragraphs give a brief overview of the history of logical neural networks and sigma–pi (probabilistic) neural networks.

5.1. Introduction to Sigma–Pi (Probabilistic) Neural Networks

The generalisation of the logical node to the multi-level probabilistic unit (Myers, 1989) is now presented below.

The diagram depicted in Figure 7 shows a simple three-state probabilistic logic node (PLN) (Kan & Aleksander, 1987; Aleksander, 1989a, b; Neville & Stonham, 1992, 1993b). Figure 7 introduces the concept of a site containing a probability value that defines the output, $P(Y=1|\mu) = S_\mu$. For a three-state unit (one should note that the term "state" implies the number of discrete states the site-value of a unit may take) the site-values are $S_\mu \in \{0, u, 1\}$ where

$$P(Y=1|S_\mu = 0) = 0$$
$$P(Y=1|S_\mu = u) = 0.5$$
$$P(Y=1|S_\mu = 1) = 1$$

defines the three possible states per site, of the basic probabilistic logic node.

This may be represented in hardware terms as a storage location (in the case of the three-state unit) in a RAM which stores the site-value, S_μ. Then $S_\mu \in \{0, 0.5, 1\}$ in binary representation is $S_{\mu_{binary}} \in \{00_2, 01_2, 10_2\}$, i.e., three values, note later in the article we term this the machine-quantised representation and designate it S_{m-q}.

We can now generalise the three-state unit to a multi-level logical node, e.g., for a five-state unit the site-value S_μ takes the following probabilities {0, 0.25, 0.5, 0.75, 1} of outputting a logical "1" if the output function is linear. If we interpret the site-value

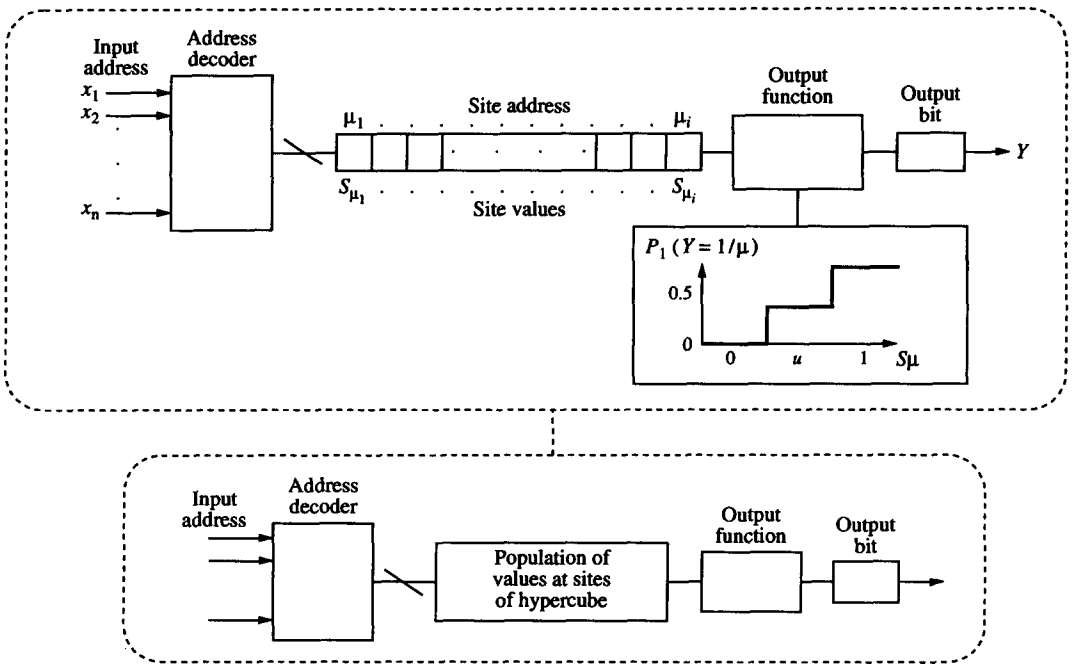

FIGURE 7. The simple three-state probabilistic logic node.

as ranging over a set of discrete levels, $S_m \in \{-2, \ldots, +2\}$, where S_m is represented in polarized notation, the output functions for the linear and sigmoidal cases is depicted in Figure 8.

The more important issue is that these multi-level units have had their functionality described mathematically by Gurney (1989, 1992a, b, 1993), they are then termed sigma–pi units (the notation is described in the following paragraphs). The important aspect of presenting a mathematical representation, which can deal with real-valued inputs and outputs (e.g., analogue or continuous valued inputs), is that one may analytically prove learning convergence in a supervised regime, such as associative reward–penalty and back propagation of error. The mathematical foundation of the sigma–pi model is covered in the following section.

6. THE SIGMA–PI NEURON MODEL

The neuron model we utilise has previously been termed a sigma–pi unit, Figure 11 (Gurney, 1989, 1992a, b, 1993), these units are similar to pRAM units (Gorse & Taylor, 1991) and as they are RAM based they may be placed in the same category as PLN units (Aleksander, 1989a, b).

The foundation of the research in this article is based on the use of an artificial neuron model known as a sigma–pi unit. The model was derived by Gurney (1989) from a mathematical description of the

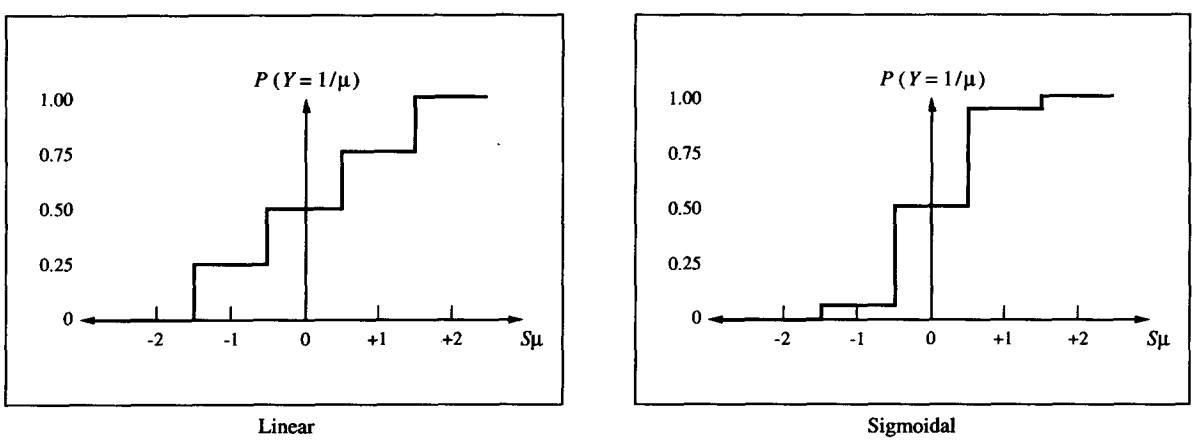

FIGURE 8. Output function five-state multi-level logic node.

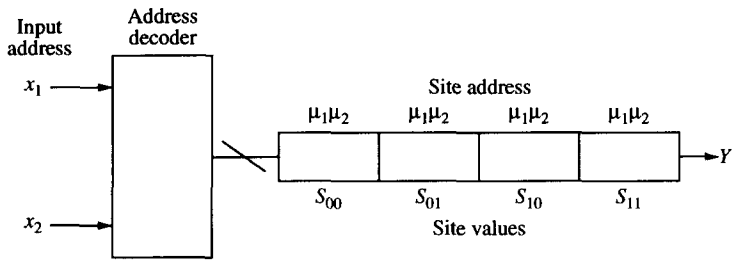

FIGURE 9. Simplistic structural details of a sigma–pi cell, for analytical visualisation.

functionality of these devices. We start by visualising a basic unit, shown in Figure 9.

Here $x \in \{x_1, x_2, \ldots, x_i\}$ is a binary input vector which may be represented as a set of bits in positions x_1 to x_i. The site address, $\mu \in \{\mu_1, \mu_2, \ldots, \mu_i\}$, is represented by a set of bits in positions μ_1 to μ_i. The site-value, S_μ, is addressed by the binary string μ.

The functionality of these sigma–pi units has been described mathematically by Gurney (1989) as

$$y = \frac{1}{2^n} \sum_{\mu} S_\mu \prod_{i=1}^{i=n} (1 + \bar{\mu}_i \bar{x}_i) \tag{11}$$

where $\bar{x}$ denotes polarised notation $\bar{x} \in \{-1, 1\}$, $\bar{\mu} \in \{-1, 1\}$ and S_μ denotes unpolarised notation, i.e., in binary $S_{\mu_{binary}} \in \{0, 1\}$. The output y is in unpolarised notation, giving a binary representation $y \in \{0, 1\}$. This may be cast in the more normal form of an activation (which in the semi-linear case is a linear sum-of-weights function) and an output function.

The activation is

$$a = \frac{1}{2^n} \sum_{\mu} \bar{S}_\mu \prod_{i=1}^{i=n} (1 + \bar{\mu}_i \bar{x}_i) \tag{12}$$

where $a = \bar{y}$, hence

$$y = \frac{1}{2}(a + 1). \tag{13}$$

One should note that the site-value is represented in its polarised form $\bar{S}_\mu$ and $\bar{y}$ which take the values $\{-1, 1\}$.

We now present the sigma–pi models, which in the case of our research take the form of stochastic models as the site-values are interpreted as probabilities. The input, x_i, may also be interpreted as the probability of a "1" appearing at the ith input to the node. The output, y, is defined by a probabilistic process which is presented in the following paragraphs.

The first model is termed a time integration node (TIN) by Gurney (1989), and contains a hypercube of sites which feeds a bit stream or activation stream which is then interpolated through an output function, the TIN is shown in Figure 10.

A time integration node stores site-values in a hypercube, which is addressed by an i bit input vector. The input vector addresses a site, μ, which contains a site-value, S_μ, which stores a value $S_\mu \in \{-S_{m,\ldots}, +S_m\}$ which we interpret as a quantised number for reasons of hardware implementa-

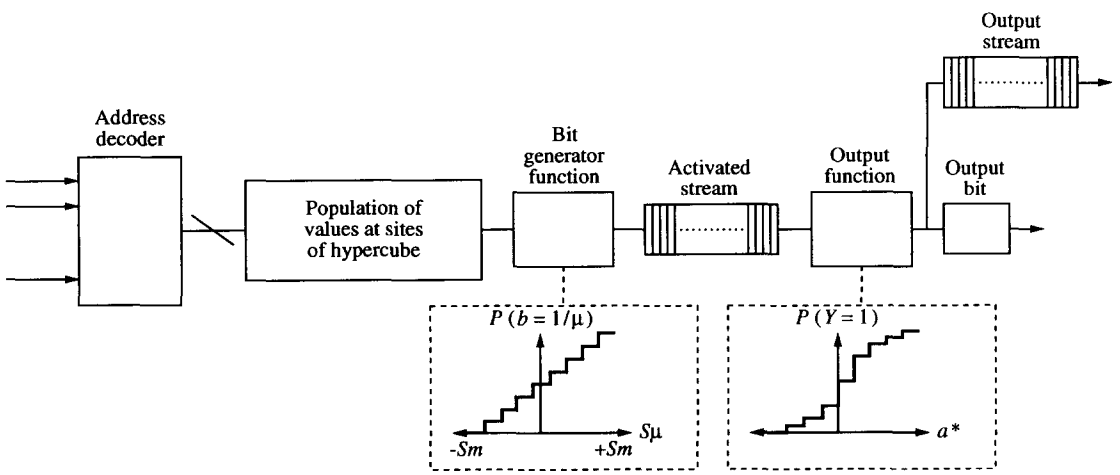

FIGURE 10. The time integration node (TIN) sigma–pi unit.

tion, but it may also be a real-valued number. The site-value is passed through a bit generator function, which fills the activation stream with a bit, "b". The stream is then used to estimate an activation value, which is then passed through an output function (which may be linear or sigmoidal) in order to produce the output.

The TIN's activation may be defined as

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

when

$$P(\mu) = \frac{1}{2^n}\prod_{i=1}^{i=n}(1 + \mu_i z_i) \tag{15}$$

given

$$P_\mu(x_i) = \frac{1}{2}(1 + \mu_i z_i) \tag{16}$$

where a is the activation which is found by summation of all the addressed sites S_μ times $P(\mu)$ the probability that the site address was visited, $S_\mu \times P_{(\mu)}$. The new input address is formed from a set of Boolean variables, $\{X_i\}$. These are defined via a set of probability distributions determined by z_i.

$$P_1(x_i) = \frac{1}{2}(1 + z_i) \quad \text{and} \quad P_0(x_i) = \frac{1}{2}(1 - z_i) \tag{17}$$

where z_i, the input probability distribution, defines the probability of the input x_i.

Hence the probability $P_\mu(x_i)$ that x_i is equal to the ith component of the site address μ is given in eqn (16), when $P(\mu)$ gives the probability that the current input address locates site μ, noting that $\langle \cdot \rangle$ denotes an expectation value and $-1 < a < 1$.

If we define the current input address as η, then use S_η to generate "a", the new activation stream bit is given by

$$P(b = 1|\eta) = \frac{1}{2}(S_\eta/S_m + 1). \tag{18}$$

Over many time steps, if the input vector is held constant

$$\langle b \rangle = P_1(b) = \frac{1}{2}\left(\frac{S_\eta}{S_m} - 1\right) \tag{19}$$

then

$$a = 2\langle b \rangle - 1 = \langle \bar{b} \rangle. \tag{20}$$

If N_1 is the number of 1s in the activation stream which is L bits long, then an estimate for $\langle b \rangle$ is $\frac{N_1}{L}$.

An estimate a^* for the activation is

$$a^* = 2\frac{N_1}{L} - 1. \tag{21}$$

This is then used to obtain an estimate, $y^* = \sigma(a^*)$, of the output y in the case of a sigmoidal output function or $y^* = (a^*)$ in the case of a linear output function. This in turn defines a distribution on a Boolean random variable y, by $P_1(y) = y^*$, which is used to communicate the output to the next layer. Under stationary conditions, the stream's output bits generated in this manner then estimate the value of "y" for this node.

The TIN may be utilised for continuous valued inputs where the real-valued input defines the probability (z_i) of entering a "1" onto an input x_i of a node. One should note in our depiction of the TIN the site-values are quantised, hence the linear bit generator is quantised into multi-levels as is the sigmoidal output function, this is due to quantisation of a^* to allow these models to be hardware realised. This method has been generalised to enable the sigma–pi units to operate with real-valued site-values, which are termed an "analogue model", we in fact utilise a unit known as a "stochastic model".

When the TIN sigma–pi unit is configured into a multi-layered net topology a change in the input probabilities means that the new outputs at the final layer will be estimated after "mL" time steps, where "m" is the number of layers in the network.

A training regime (i.e., associative reward–penalty) requires correlations between short term fluctuations in the output of the nodes in subsequent layers to provide information for training, we utilise the non-linearity $\sigma(\cdot)$ in the output stream bit generator for this reason. This may not be required if, for example, we were using a TIN to perform an approximation to a function, e.g., a polynomial function, and obviously if we required a linear unit we would utilise a linear output function.

Another type of model exists which we will term the "direct output node" (DON), as it does not utilise the bit stream to provide an output. With this the activation in the analogue model is

$$a = \frac{1}{2^n}\sum_{\mu}\sigma(S_\mu)\prod_{i=1}^{i=n}(1 + \bar{\mu}_i z_i) \tag{22}$$

so that in the stochastic model

$$a = \sum_{\mu}\sigma(S_\mu)P(\mu) = \langle \sigma(S_\mu) \rangle. \tag{23}$$

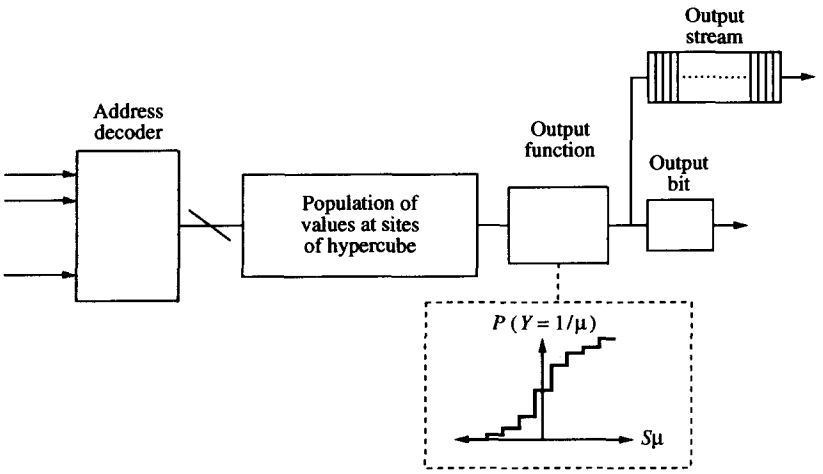

FIGURE 11. The direct output node (DON) sigma–pi unit.

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

$$P(y = 1|\eta) = \sigma(S_\mu). \tag{24}$$

Hence in the TIN's case $y = \sigma(\langle S_\mu / S_m \rangle)$ whereas in the DON $y = \langle \sigma(S_\mu) \rangle$. The stochastic model of the DON is depicted in Figure 11. If one requires a linear interpretation of the output the σ notation may be dropped.

The sigmoidal function $\sigma(S_\mu)$ is defined as

$$\sigma(S_\mu) = \frac{1}{1 + e^{-S_\mu/\rho}} \tag{25}$$

where ρ is a positive valued parameter that determines the shape of the curve, re. the start of Section 5.

The DON is a simplified version of a TIN with no activation stream, so the site-values are only processed through the output functions to any subsequent layers and output streams. The site-values in the DON are quantised and if the output stream is not required as a stochastic representation of the nodes' output (activation), then the resultant model is the same as the multi-level probabilistic logic node (MPLN) of Myers and Aleksander (1988).

In our research we use the stochastic model direct output node (DON). The output behaviour of these units is similar to that of the Boltzmann units of Hinton et al. (1984, 1985). The direct output node is so called as it produces an output directly from the site-value stored at the site-address defined by the input vector and not from an activation stream which the TIN utilises to obtain an output.

6.1. The Quantised Model and Low-Level Machine Representation

For our implementation all site-values are quantised, denoted "q". This enables the sigma–pi unit to be hardware realised, hence the site-value is divided into D discrete levels, where

$$\bar{S}_{\mu_q} \in \{-S_m, \ldots, +S_m\} \tag{26}$$

and, hence, there are

$$D = (2S_m + 1) \tag{27}$$

levels.

This leads to the interpretation of $\bar{S}_\mu$ as equal to

$$\bar{S}_\mu = \frac{\bar{S}_{\mu_q}}{S_m} \tag{28}$$

then

$$\bar{S}_{\mu_q} = \{n|n = -S_m, \ldots, -1, 0, +1, \ldots, +S_m\} \tag{29}$$

hence if $S_m = 5$ then

$$\bar{S}_{\mu_q} = \{n|n = -5, \ldots, -1, 0, +1, \ldots, +5\} \tag{30}$$

giving

$$\bar{S}_{\mu_q} = \left(\frac{S_{\mu_q}}{S_m}\right) = \{n'|n' = -1, \ldots, -0.2, 0, +0.2, \ldots, +1\}. \tag{31}$$

This may be represented in hardware as:

$$S_{\mu_{m-q}} = \{n''|n'' = 0, \ldots, S_m, \ldots, 2S_m\} \tag{32}$$

which we term the machine-quantised $(m-q)$ representations, which is normally stored in binary form, which for $S_m = 5$ is

$$S_{\mu_{m-q}} = \{n'' | n'' = 0000_2, \ldots, 0101_2, \ldots, 1010_2\} \quad (33)$$

which is the low-level machine representation of the site-value.

6.2. Associative Reward–Penalty for Sigma–Pi Units

In order to define a reward–penalty signal "r" [refer to Section 3.3, eqn (5)] for sigma–pi units, we first introduce the mean-squared output error term:

$$e_0 = \frac{1}{N_v} \sum_{i=1}^{i=I_v} \left[Y_t^i - \sigma\left(S_\mu^i\right) \right]^2 \quad (34)$$

where $[\cdot]^2$ is the square error per input stimuli, defined on the output—this is summed over all N_v output units or visible units. The sum over the set I_v of these output units. The error is the difference between the target response, $Y_t^i \in \{0, 1\}$, of a stated output for a given input/output pattern pair and the sigmoidal value of the site $\sigma\left(S_\mu^i\right)$, where μ specifies an address on unit i's input lines. The scalar reward is defined probabilistically by eqn (5).

The site-values of the sigma–pi units are adapted dependent on the reinforcement value, r, for each node, j, for input address μ then

$$\Delta S_\mu^i = \begin{cases} \alpha[Y^j - \sigma(S_\mu^j)] & \text{if } r = 1 \\ \alpha\lambda_{rp}[1 - Y^j - \sigma(S_\mu^j)] & \text{if } r = 0 \end{cases} \quad (35)$$

where $\alpha > 0$ is the learning rate and $0 \geqslant \lambda_{rp} \geqslant 1$ is the penalty coefficient.

6.3. Training Output Units

The output units are trained using a delta rule, where the addressed site, μ, of output node, i is updated using:

$$\Delta S_\mu^i = \begin{cases} \alpha[Y^i - \sigma(S_\mu^i)] & \text{if } r^i = 1 \\ \alpha\lambda_{rp}[1 - Y^i - \sigma(S_\mu^i)] & \text{if } r^i = 0 \end{cases} \quad (36)$$

where the reward signal "r^i" for each output unit i is derived from the unit's output error

$$e_0^i = \left[Y_t^i - \sigma\left(S_\mu^i\right) \right]^2 \quad (37)$$

where $Y_t^i \in \{0, 1\}$ is the target response for the unit. A different approach is to use the delta rule of Widrow and Hoff (1960), where the output units are adapted with:

$$\Delta S_\mu^i = \alpha\rho \frac{2}{N_v} \sigma'\left(S_\mu^i\right)\left[Y_t^i - \sigma\left(S_\mu^i\right) \right]. \quad (38)$$

The adaption technique above is a symmetrical learning rule, our methodology utilises a non-symmetrical learning rule for updating of visible and hidden units, as the noise term λ is then utilised to avoid absorption to sub-optimal states. This philosophy is also followed by Gorse and Taylor (1990a, b), but they utilise separate independent reward and penalty signals.

6.4. Real-Valued Associative Reward–Penalty $A_{R-P}^{\Re}$

In the preceding section we have presented the standard A_{R-P} training regime which utilises a binary reinforcement signal "$r \in \{0, 1\}$". In order to evaluate the effects of enlargement of the reinforcement signals bandwidth, we utilise the real-valued reinforcement training regime of Barto et al. (1987). This method is utilised to enable us to compare the real-valued Associate Reward–Penalty training rule with the quantised Adaptive Critic methodology presented in the preceding paragraphs.

The original Associative Reward–Penalty of Barto (1987) utilises a scalar reinforcement signal, $r \in \{0, 1\}$, defined in (5) which is probabilistically interpreted from the output error. Barto and Jordan (1987) call this type of regime the "P-model" A_{R-P} rule. To enlarge the bandwidth of the information globally broadcast to a network, a real-valued reinforcement was utilised $0.0 \leqslant r^{\Re} \leqslant 1.0$ given by

$$r^{\Re} = 1 - e_0 \quad (39)$$

where e_0 is defined in eqn (34). Barto and Jordan (1987) calls this type of regime the "S-model" A_{R-P} rule.

The real-valued reward–penalty $A_{R-P}^{\Re}$ algorithm for each node j, for input address μ now becomes:

$$\Delta S_\mu^i = \alpha\left[Y^j - \sigma\left(S_\mu^j\right) \right] r^{\Re} + \alpha\lambda_{rp}\left[1 - Y^j - \sigma\left(S_\mu^j\right) \right](1 - r^{\Re}). \quad (40)$$

The $A_{R-P}^{\Re}$ rule is non-symmetrical update rule if $0.0 \leqslant \lambda_{rp} \leqslant 1.0$, which allows both reinforcement and penalisation to affect the update process simultaneously.

6.5. Quantised Sigma–Pi Model Training and Noise

In order to approximate the expectation of ΔS_μ over time, the fractional component of ΔS_μ is interpreted as a probabilistic component. Then the site changes in the quantised model are:

$$\Delta_q S_\mu = I_u + F_u \tag{41}$$

where I_u is the integral part of the left hand side (LHS), formed by truncating the Δ change in the site-value; F_u is the remaining fractional part.

Then

$$\Delta_q S_\mu = I_u + \delta S_\mu \tag{42}$$

where δS_μ is a unit increment or decrement made stochastically with probability $|F_\mu|$. That is

$$\begin{aligned} P(\delta S_\mu = +1) &= \frac{1}{2}[1 + sgn(\Delta S_\mu)]|F| \\ P(\delta S_\mu = -1) &= \frac{1}{2}[1 - sgn(\Delta S_\mu)]|F| \\ \delta S_\mu &= 0 \ otherwise \end{aligned} \tag{43}$$

where F is the fractional component and "*sgn*" is the sign of the delta change ΔS_μ (i.e., if $\Delta S_\mu = 1.25$ then $sgn(\Delta S_\mu) = +1$ and if $\Delta S_\mu = -1.6$ then $sgn(\Delta S_\mu) = -1$).

This gives the required expectation

$$\langle \Delta_q S_\mu \rangle = S_\mu \text{ or } \Delta_q S_\mu = \Delta S_\mu + n_q \tag{44}$$

where n_q is a noise term with zero expectation. We observe that the effective noise n_q may be increased or decreased dependent on the value of S_m and ρ [i.e., eqn (25)]. When $S_m \to \infty$ the noise term $n_q \to 0$, but by selection of an $S_m = 10$, say, we may obtain beneficial results from the noise term.

We may also set the slope of the sigmoid, which previous researchers have set to a high slope $\rho = 0.04$ (Penny et al., 1990), to a semi-linear curve, i.e., $\rho = 0.3$ and hence more noise may also be introduced into the incremental learning regime. This hypothesis has been extended in the light of Gullapalli's (1988) work, as he postulates that by keeping the values of the units output, Y, from saturating (i.e., going to their maximum values, as in the case of Y pertaining to a real number) one may enhance learning efficiency. In our case by using $\rho = 0.3$ one does not allow the $\sigma(\cdot)$ to asymptotically reach its maximum value, hence the difference term $[Y - \sigma(S_\mu)]$ never goes to zero (which it would do if $\rho = 0.04$, where it has a very narrow non-asymptotic region). One should note by setting $\rho = 0.3$ we obtain a probabilistic output even when the site-value reaches its extreme asymptotic values. This may also be another reason why we obtain more efficient learning when $\rho = 0.3$. Initial studies of the variation of S_m and ρ were carried out by Myers (1989), where a value of $S_m = 5$ ($D = 11$), gave optimal results when tested on seven-bit parity and simple 16-pattern generalisation tests.

The reader should of course be aware that functions (35) and (36) will be clipped at their extreme values due to the nature of the bounding of $S_\mu \in \{-S_m, S_m\}$, but this may be overcome if one multiplies the whole of the right hand side (RHS) by sigma primed.

6.6. Algorithmics of the A_{R-P} Training of Sigma–Pi Networks

The algorithm presented in pseudo-code for associative reward–penalty training of sigma–pi units is detailed below.

```
clamp training vector
  for each layer do {
    latch addresses
    generate new site-values
    generate new output bits
  } od
calculate error e0
generate reinforcement bit r
  for each unit in each layer do {
    calculate ΔSμ
  } od
update site-values.
```

Making error estimates with eqn (34), on the ascent of r [defined in Section 3.3, eqn (5)], then using eqn (35) for the hidden units and eqn (36) for the visible or output units updates.

One should note that this is a "sequential" training method, where each vector is presented, delta changes in the site-values ΔS_μ are made, then the next vector is presented. This differs from the methodology used by Rumelhart et al. (1986) who use a "batched" training update method, whereas we only require errors on a per pattern basis or "sequential" training update method.

7. THE QUANTISED ADAPTIVE CRITIC ELEMENT

In the subsequent work on the adaptive critic, we utilise the formalised conventions which we have previously used for the sigma–pi model, refer to Sections 6 and 6.1, which are:

(1) all stimuli to the action net and the adaptive critic element are binary,

(2) the binary input vectors, which stimulate the critic, address site locations,
(3) the site-values stored at the locations may be viewed as values stored in n-tuples,
(4) the stored site-values are quantised.

The following paragraphs formally introduce our adaptive critic methodology, in order to utilise the sigma–pi convention of notation, for tasks requiring temporal predictions such as control systems or sequential tasks that require temporal information (Barto et al., 1983; Myers, 1990). Then we put forward an adaptive critic element (ACE) for the more orthodox tasks, such as those carried out by feedforward networks. Werbos (1990) previously stipulated that the adaptive critic can be used to adapt conventional networks that perform tasks like pattern recognition. In the succeeding work we extend Barto's (1987) associative reward–penalty (A_{R-P}) paradigm to incorporate Barto et al.'s (1983) research and introduce an adaptive critic which is connected to a sigma–pi network (action network), in order to aid the action network while it is being trained.

7.1. The Quantised Adaptive Critic for Sigma–Pi Networks

We now formulate the mathematical notation of an adaptive critic for sigma–pi nets.

The adaptive critic element receives an external reinforcement, as used in the standard A_{R-P} (re. Section 3), derived from an action network's output (re. Section 3.3), which is a scalar signal.

In order to evaluate the internal reinforcement, $\hat{r}_{(t)}$, we calculate:

$$\hat{r}_{(t)} = r'_{(t)} + \gamma P_{(t)} - P_{(t-1)}. \tag{45}$$

The prediction, $P_{(t)}$, relates to the present prediction while $P_{(t-1)}$ is the past prediction—these values are stored in a quantised manner, where $P_{(\cdot)} \in \{0, \ldots, +P_n\}$, giving $D = P_{n+1}$ discrete levels, which are stored as q bit numbers (e.g., if $P_n = 8$ then $P_{(\cdot)} \in \{0.125n | n = 0, 1, \ldots, N\}$ where $N = 0.125/0.125 = P_n$). The multiplying coefficient $0.0 < \gamma \leqslant 1.0$ has previously been termed the "discount factor", which in our case may be thought of as a means of reducing the effect the most recent prediction has on the evaluation of $\hat{r}_{(t)}$, if $\gamma < 1.0$. The reinforcement value $r' \in \{-1, +1\}$ denotes a scaled reward [re. Section 7.2, eqn (48)], which enables the predictions, $P_{(\cdot)}$, to incrementally increase from zero to one and also be decremented from one back down to zero. Note, to use $\hat{r}_{(t)}$ with A_{R-P} we must re-scale it [c.f. eqn (52) in Section 7.2].

The prediction is updated by:

$$\Delta P_{(t+1)} = \beta \hat{r}_{(t)} \bar{x}_{v(t)} \tag{46}$$

where $0 < \beta < 1$ is a rate coefficient which defines by how much the prediction is incremented or decremented. The input eligibility, $\bar{x}_{v(t)}$, is interpreted as; given an "i" bit input vector $\{x_1, x_2, \ldots, x_i\}$, which addresses location "v" in an eligibility n-tuple, giving an eligibility value $\bar{x}_{v(t)} \in \{0, \ldots, +\bar{x}_n\}$, that is specified as a "q" bit number, having $D = \bar{x}_n + 1$ discrete quantisation levels (e.g., if $\bar{x}_n = 16$) then $\bar{x}_{v(\cdot)} = \{0.0625n | n = 0, 1, \ldots, N\}$ where $N = 1.0/0.0625 = \bar{x}_n$.

The input eligibility trace is updated by:

$$\bar{x}_{u(t+1)} = \lambda_{ace} \bar{x}_{u(t)} + (1 - \lambda_{ace}) x_{v(t)} \tag{47}$$

for all input addresses $0 \leqslant u \leqslant \eta$, where η equals the maximum input address (e.g., for an 8-tuple $\eta = (2^8) - 1$ or 255 decimal or FF hexidecimal), and where $x_{v(t)}$ is a binary trigger for the eligibility trace, when site "v" is addressed $x_{v(t)} = 1$ (i.e., where input address v equals the n-tuple address u) and all other non-addressed traces are updated with $x_{u \neq v(t)} = 0$, and where $0 < \lambda_{ace} < 1$ is a positive constant determining the decay rate. The input eligibility figures are now viewed as quantised values stored as a set of traces in a n-tuple, which are updated every time the action network carries out a "forward pass" operation.

The above formalises the adaptive critic to what we will term the "quantised adaptive critic element" (QACE). However, the above quantised critic is usually utilised for tasks where temporal information is being used. The paragraphs below instruct one how the QACE rule may be amended, to utilise the QACE in the more normal "static" mapping situations.

7.2. Quantised Adaptive Critic Element for Pattern Mapping Tasks

The QACE we define here carries out static mapping tasks, which is one that maps input vector X to output vector Y, where Y belongs to one of k different classes. The quantised adaptive critic stores the variables that it utilises in the same manner as the

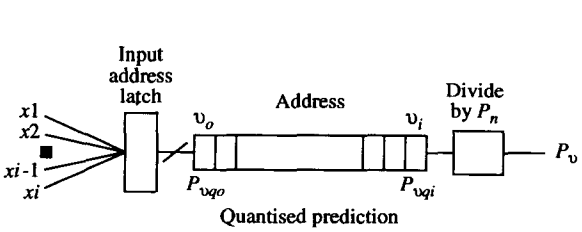

FIGURE 12. Diagrammatic representation of the prediction values stored in an n-tuple.

sigma–pi unit (re. Section 6.1). The QACE is addressed by an i bit vector, defined by the input vector $\{x_1, x_2, \ldots, x_i\}$, which addresses a location defined as address v.

The vector v addresses three locations in the ACE in parallel, these site-values store the prediction values $P_{v(t)}$, $P_{v(t-1)}$ and $\bar{x}_{v(t)}$ the eligibility values, which are all q bit numbers, of the type defined in Sections 6.1 and 7.1. The method of storing the predictions (which is also utilised for storage of the eligibility values) is diagrammatically depicted in Figure 12.

The QACE utilises the standard external reward, $r_{(t)}$, which is given in eqn (5), Section 3.3, where $r_{(t)} \in \{0, 1\}$ is a binary scalar value. The reinforcement is then scaled, to enable one to obtain a reinforcement which may be a negative -1 or positive $+1$ value, in order to calculate a prediction of the reinforcement ΔP_v value that increases and decreases, the scaled reinforcement is given by:

$$r'_{(t)} = (2 \times r_{(t)}) - 1. \tag{48}$$

We then obtain a bivalent reinforcement of the type used by Barto et al. (1983), where $r' \in \{-1, +1\}$. The scale reward signal is then used to derive an improved or internal reinforcement signal, given by:

$$\hat{r}_{(t)} = r'_{(t)} + \gamma P_{v(t)} - P_{v(t-1)} \tag{49}$$

where $P_{v(t)}$ is the present prediction and $P_{v(t-1)}$ the past prediction. It should be noted that this is not the same as Barto's (1983) original work, where he uses the prediction values $P_{(t)}$ and $P_{(t-1)}$. We use the present and previous prediction values for the given site address v. The multiplying coefficient $0.0 < \gamma \leqslant 1.0$ has previously been termed the "discount factor", re. Section 4.1.

The prediction value is updated by:

$$\Delta P_{v(t+1)} = \beta \hat{r}_{(t)} \bar{x}_{v(t)} \tag{50}$$

where $0 < \beta < 1$ is a positive constant determining the rate of change of $P_{v(\cdot)}$. All the input eligibility traces are updated using:

$$\bar{x}_{u(t+1)} = \lambda_{ace} \bar{x}_{u(t)} + (1 - \lambda_{ace}) x_{v(t)} \tag{51}$$

for all input addresses $0 \leqslant u \leqslant \eta$, where η = the maximum input address (e.g., for an 8-tuple $\eta = (2^8) - 1$ or 255 decimal or FF hexidecimal), and where λ_{ace}, $0 < \lambda_{ace} < 1$ determines the eligibility trace's decay rate.

The binary value, x_v, is a trigger for the eligibility trace, and when the site v is addressed $x_v = 1$ and all other non-addressed traces are updated with $x_{u \neq v} = 0$. The internal reinforcement, $\hat{r}_{(t)}$, is then re-scaled

$$r^*_{(t)} = \frac{1}{2}(\hat{r}_{(t)} + 1.0) \tag{52}$$

which denotes a quantised reinforcement $r^*_{(t)} \in \{0.0, \ldots, 1.0\}$, that is termed the "bounded" reward signal $r^*_{(t)}$ [e.g., $r^*_{(t)} \in \{0.125n | n = 0, 1., \ldots, N\}$ for all our experiments $P_n = \bar{x}_n = N$, we set N equal to 8]. It is termed "bounded" reinforcement signal as the re-scaled reward (52) is limited to the maximum and minimum values of Barto's (1987) scalar reinforcement (i.e., minimum = 0 and maximum = 1). When the internal reinforcement is not limited to these extremes it is then defined as an "unbounded" reinforcement signal, $r^*_{(t)}$, which denotes a quantised reinforcement $r^*_{(t)} \in \{-1.0, \ldots, +2.0\}$, which permits penalisation even when the penalty coefficient, λ_{rp}, in the associative reward penalty training regime is zero, i.e., $\lambda_{rp} = 0$.

The sigma–pi unit's addressed sites, given node j and site address μ, are then updated using:

$$\begin{aligned} \Delta S^j_\mu = {} & \alpha \left[Y^j - \sigma\left(S^j_\mu\right) \right] r^*_{(t)} \\ & + \alpha \lambda_{rp} \left[1 - Y^j - \sigma\left(S^j_\mu\right) \right] \left(1 - r^*_{(t)} \right). \end{aligned} \tag{53}$$

The reader should of course be aware that the function (53) will be clipped at its extreme values due to the nature of the bounding of $S_\mu \in \{-S_m, \ldots, S_m\}$, but this may be overcome if one multiplies the whole of the right hand side (RHS) by sigma primed.

7.3. Example Evaluation of the Internal Reward Using the Adaptive Critic Elements

The ACE methodology utilises one ACE per net, as per the Barto et al. (1987) initial research. This is based on the premise that the ACE is utilised to predict "r", which is derived from the input eligibilities obtained from the environment R.

In Figure 13 we present a diagrammatic representation of an example evaluation of the internal reward signal using the ACE. In time period t the external reward, $r_{(t)}$, the past $P_{v(t-1)}$ and present $P_{v(t)}$ predictions are $r_{(t)} = 0$ and $P_{v(t)} > P_{v(t-1)}$. Then the internal reward, $\hat{r}_{(t)}$, defined by eqn (49) calculates a pseudo-penalty value (i.e., the internal reward is not equal to minus one) which means that the internal re-scaled reward $r^*_{(t)}$ is a non-zero reinforcement signal. The effect the ACE has in this case is to reduce the penalty signal to the artificial neural network that it is advising. This was caused by the present prediction, $P_{v(t)}$, being larger than the past prediction, $P_{v(t-1)}$.

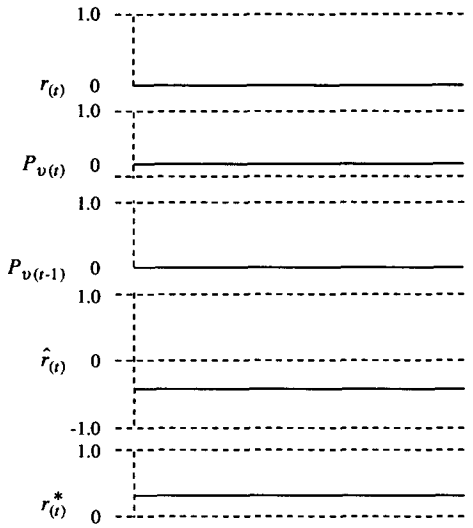

FIGURE 13. Example evaluation of the internal reward.

7.4. Multi-Cube Feedforward Structures of Sigma–Pi Units

The sigma–pi model we use may be defined as a collection of site-values at the corners of a hypercube and as such are termed cube based nodes. One of the problems with fully connected networks of sigma–pi (logical) nodes that require each unit to cover a large input retina is that these units suffer from the problem of exponential rise in resources as the number of inputs increases. For example if one uses a 3-tuple we require $2^3 \rightarrow 8$ sites in the cube. But if we need a fully connected net to cover an input retina of 25 inputs, then we require a 25-tuple giving $2^{25} \rightarrow$ approx. 33×10^6 sites in the cube.

A single cube nodes functionality may contain thousands more functions than required. If one considers the standard sigma–pi unit with a single cube of site-values followed by an output function, a linear extension of this type of structure would be to sum the outputs from several cubes (Gurney, 1989) and then pass this through an activation–output function. This is depicted in Figure 14.

The next extension to this type of structure would be to utilise linear weights on connections from other inputs or units to the summation unit of the multi-cube unit, to enable these units to be configured into competitive networks (i.e., in competitive nets, the units compete for the opportunity to respond to the input stimuli). One should note that the multi-cube structure with no output function, but just an integer output, is the same structure as the WISARD system (Wilkie, 1983), whereas if one configures the multi-cube unit with a hard limiting (or threshold) output function we have the type of topology put forward by Minsky and Papert (1969) for their perceptron.

The multi-cube structure overcomes the restriction that single-cube sigma–pi units have, as the multi-cube is a linearly scaleable unit. We may also generalise this methodology to store other variables (not just site-values), as we have done in our work on the quantised adaptive critic (Neville, 1993; Neville & Stonham, 1993, 1994b, c) to overcome general problems of exponential increase of resources, which relate to the number of inputs to a system.

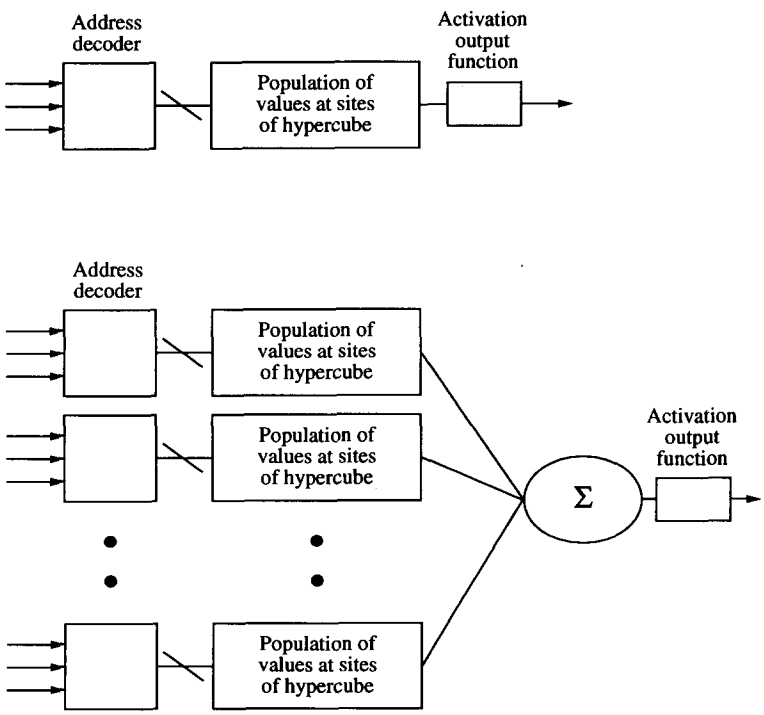

FIGURE 14. Single-cube sigma–pi unit TOP and a multi-cube sigma–pi unit BOTTOM.

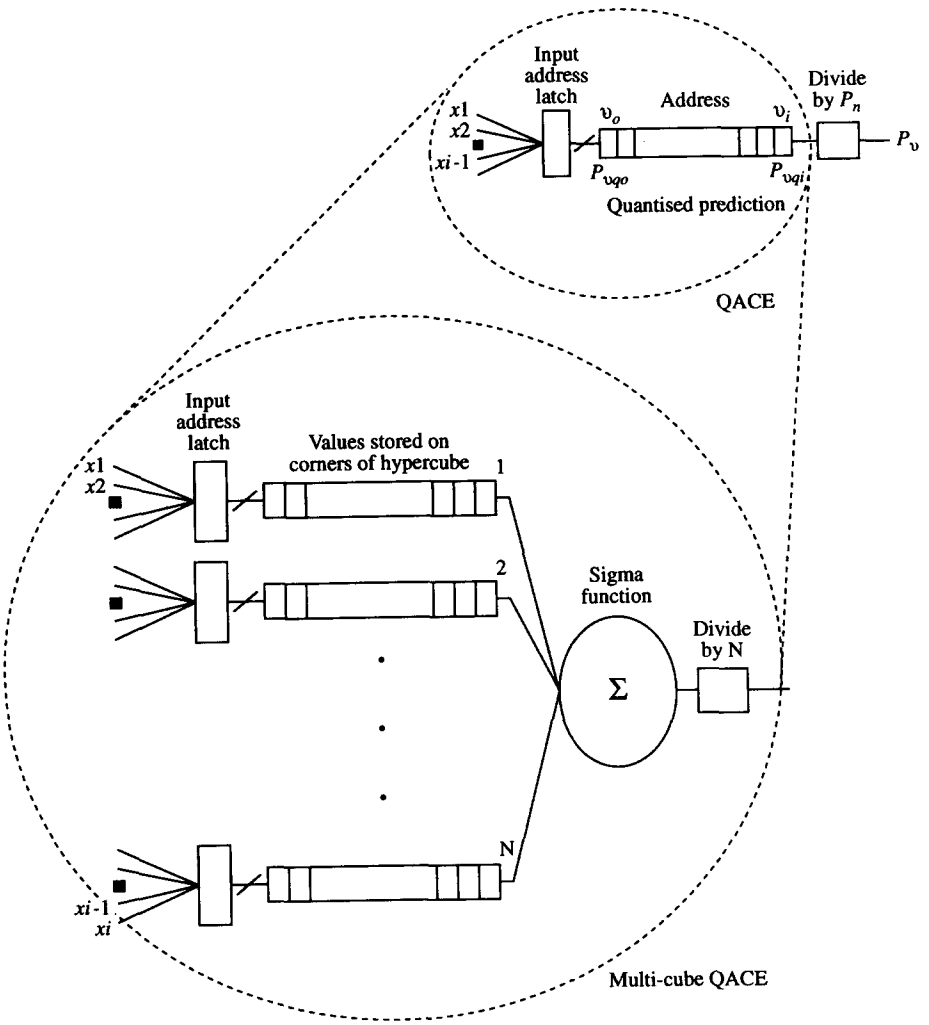

FIGURE 15. Multi-cube quantised adaptive critic element.

7.5. The Multi-Cube QACE

The basic adaptive critic methodology utilises a single cube or n-tuple (where n is the number of inputs to the QACE, giving full coverage of the input retina), for each of the $P_{v(t)}$, $P_{v(t-1)}$ and $\bar{x}_{v(t)}$ variables.

If the ACE has to operate with a large number of input variables, then using a single n-tuple to cover the whole input space would not be feasible as the storage requirements grow exponentially. For example, if we had 20 input lines (variables) we would require 2^{20} storage locations for each of the three different stored variables (i.e., the $P_{v(t)}$, $P_{v(t-1)}$ and $\bar{x}_{v(t)}$). The multi-cube QACE is utilised to overcome the exponential rise of resources as the number of input variables rise. If one considers the QACE's input lines as connected to an input retina, then the input retina is sub-divided into sub-retinas.

Each of the partial retinas is then allocated three tuples to store the $P_{v(t)}$, $P_{v(t-1)}$ and $\bar{x}_{v(t)}$ variables for that region. Then to calculate the overall or mean prediction, one sums all the partial predictions and divides by the number of n-tuples (sub-cubes) covering the whole input retina.

The multi-cube QACE, Figure 15, overcomes the above restrictions on its memory requirements as it is linearly scalable and may be utilised when one requires full connectivity. The mean output of these multi-cube structures is derived from several cubes, termed sub-cubes.

The internal reinforcement then becomes:

$$\hat{r}_{(t)} = r'_{(t)} + \gamma \bar{P}_{v(t)} - \bar{P}_{v(t-1)} \tag{54}$$

where $\bar{P}_{v(\cdot)}$ is the mean of all the addressed sites of the "N" sub-cubes.

The mean prediction, $\bar{P}_{v(\cdot)}$, given by:

$$\bar{P}_{v(\cdot)} = \frac{1}{N} \sum_{k=1}^{k=N} P_{v_k(\cdot)} \tag{55}$$

where the vectors defined by v_k, are the partial

addresses presented to the sub-cones when the input vector address is v. The prediction values are now updated using the mean eligibility, $\bar{x}_{v(t)}$, derived from all the sub-cubes, which is given by:

$$\bar{x}_{v(t)} = \frac{1}{N}\sum_{k=1}^{k=N} \bar{x}_{v_{k(t)}}. \tag{56}$$

Hence the prediction is updated using:

$$\Delta P_{v_{k(t+1)}} = \beta \hat{r}_{(t)} \bar{x}_{v(t)}. \tag{57}$$

The input eligibility traces are updated as before, on a per sub-cube basis, where each sub-cube covers a fraction of the input retina.

8. EXPERIMENTAL WORK

8.1. The 8-3-8 Encoder

The bench mark used to evaluate our research on the quantised ACE training with binary data was the 8-3-8 Encoder, Figure 16, previously utilised by Hinton et al. (1984). The encoder network has eight inputs, three hidden units and eight outputs. The encoder's task is to transmit eight-bit binary input vectors across the hidden layer boundary, hence these units must learn to represent each vector with a different three-bit code. The problem relates to the complexity of coding training data onto the narrow channel. To explain this, we look at the case when only one of the eight bits in the input vector has a set bit, then the hyperplane in each output unit separates off a single corner on the 3-cube unit. This may be coded in eight ways, one for each corner of the cube, for the first output unit, leaving seven different coding choices for the next unit and so on giving 8 different coding solutions. This means there are 8^8 possible code sets, so that only approximately 0.24% of these are useful. But to make the coding task hard we have four adjacent set-bits. Now each hyperplane must separate the cube into two equal parts. But when this is done by one node, there are immediate constraints on the coding the neighbouring units may utilise. Once four nodes have been specified the others are uniquely determined. This coding scheme yields 192 viable valid codes, which represents about 0.0011% of all possible code solutions.

Since all simulations begin with all site-values $\sigma(S_\mu) = 0.5$ or $S_{\mu_q} = 0$, giving $P(Y = 1|\mu) = 0.5$, i.e., 50% probability of the output Y obtaining a value "1", in other words no prior information has been bestowed on the network, then finding a solution to such a problem requires that the two visible groups come to agree upon the meaning of a set of codes without any prior conventions for communicating through the hidden units.

8.2. The 8-3-8 Encoder and Experimental Delimitations

The encoder we utilise for these experiments is an 8 input, 3 hidden unit and 8 output unit network as described in Section 8.1. The 8-3-8 encoder was used as a benchmark for this work. The training vector set used were hexadecimal numbers $\{F0, 78, 3C, 1E, 0F, 87, C3, E1\}$, hence $N_p = 8$. The training set was randomly ordered for each sample and a different seed was given to the stochastic operator of the net at the start of each training session. The training vectors each have four adjacent set-bits.

The output-stream of the direct output node (DON) is utilised in the experimental work to communicate an estimate of the output value,

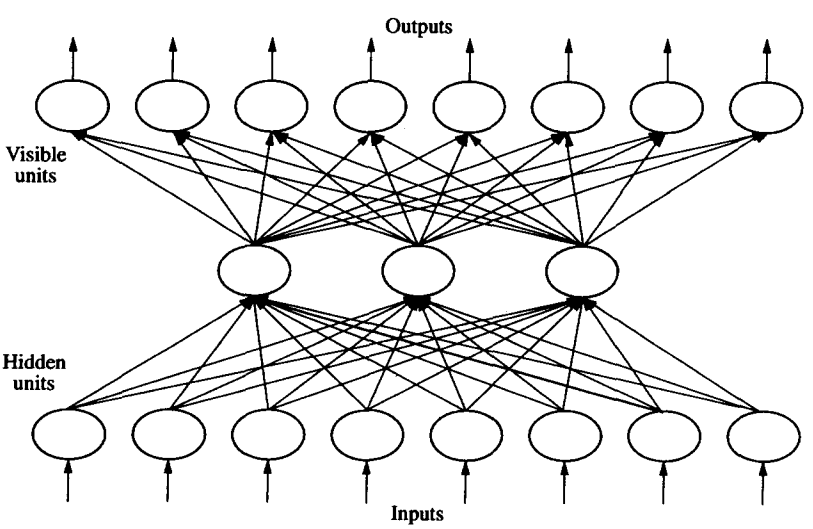

FIGURE 16. The 8-3-8 encoder network.

$y^* \in [0,1]$. The output-stream is utilised in the following manner. The current input address μ addresses, S_μ, which is used to generate a new output-stream bit. We now count the number of 1s (defined as N_1) in the output-stream. Then an estimate for y^* is obtained using:

$$y^* = \frac{N_1}{L} \tag{58}$$

where L is the length of the output-stream and N_1 is the number of ones in the stream. Assuming stationary conditions, the stream of output bits generated in this manner will, over time, transmit an estimate of the value of y for a given node. The stream of bits appearing at the output may be likened to the trains of action potentials transmitted along axons. This means one is using a time integration of the output-stream to derive an estimate of the output's value.

In order to define our error metric we first define a mean-squared error over a set of N_p patterns as:

$$E = \frac{1}{N_p} \sum_{k=1}^{k=N_p} e_0' \tag{59}$$

where the mean-squared error, e_0', is

$$e_0' = \frac{1}{N_v} \sum_{i=1}^{j=I_v} (y_t^i - y^{i*})^2 \tag{60}$$

where $Y_t^i \in \{0,1\}$ was the target output and y^{i*} was estimated from an output-stream which was a 100 bit long stream, $L = 100$, this is summed over all N_v output units. The sum is over the set I_v of these output units. The results presented in the following paragraphs show graphs of the error $\bar{e}_0$, given by:

$$\bar{e}_0 = \frac{1}{N_{tn}} \sum_{n=1}^{n=N_{tn}} E \tag{61}$$

the average error over one hundred trained networks ($N_{tn} = 100$) after 6000 training cycles had elapsed, over all N_p training patterns, where each output, y^*, is calculated after 100 forward passes. It is important to note that if this is not done and one takes the instantaneous output of the sigma–pi visible units one obtains a probabilistic output and we require an estimate of a deterministic output.

For all experiments the constants we utilise are $\rho = 0.3$, $\lambda_{rp} = 0.0$ and $S_m \in \{-10, \ldots, +10\}$. For the ACE $\gamma = 0.95$, $\lambda_{ace} = 0.8$, $\beta = 0.5$ and $P_n = \bar{x}_n = N = 8$. Each graph has a y-axis which is the average error, $\bar{e}_0$, and an x-axis which is partitioned for eight learning rates $\alpha = \{0.1, 0.25, 0.5, 1.0, 2.0, 5.0, 10.0, 20.0\}$.

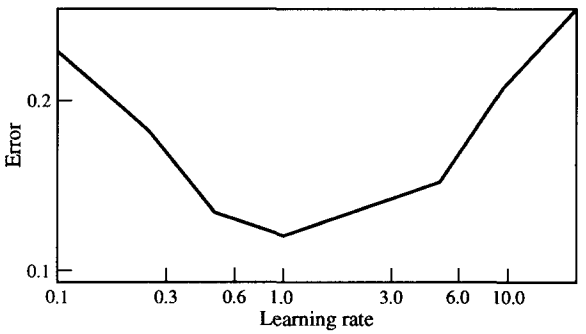

FIGURE 17. Average log error $\bar{e}_0$ versus log α, for sigma–pi based 8-3-8 encoder with eight training vectors having four set-bits. The graph shows the average error $\bar{e}_0$, over 100 trained nets, after the nets have been trained for 6000 cycles.

8.3 Standard Associative Reward–Penalty A_{A-P}

Presented in Figure 17, is a simulation of the effect of varying the standard associative reward–penalty's learning rate α, as previously described in Section 8.2. The graph shows that the optimal learning rate is $\alpha = 1.0$ as it reduces the average error, $\bar{e}_0$, by the largest amount. For learning rates les than or greater than $\alpha = 1.0$, one obtains a lower average error figure. This graph is presented as a base line for comparing the results presented in this section.

8.4. Real-Valued Reinforcement Associative Reward–Penalty $A_{R-P}{}^{\Re}$

Comparison of the standard associative A_{R-P} and real-valued $r^{\Re}$, $A_{R-P}{}^{\Re}$ is shown in Figure 18. The results presented in the graph show that for learning rates greater than $\alpha = 2.0$ the real-valued reinforcement associative reward–penalty was more efficient at

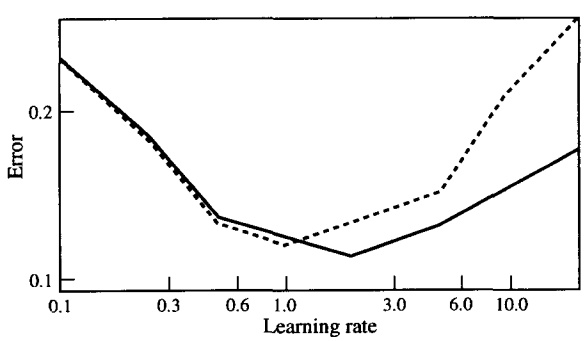

FIGURE 18. Average log error $\bar{e}_0$ versus log α, for sigma–pi based 8-3-8 encoder with eight training vectors having four set-bits. The graph shows the average error $\bar{e}_0$ over 100 trained nets, after the nets have been trained for 6000 cycles. The dotted line shows the error for the standard A_{R-P}. The solid line shows the error with real-valued reinforcement $r^{\Re} \in [0,1]$.

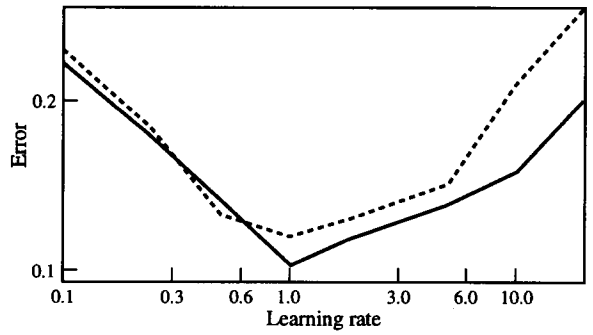

FIGURE 19. Average log error $\bar{e}_0$ versus log α, for sigma–pi based 8-3-8 encoder with eight training vectors having four set-bits. The graph shows the average error $\bar{e}_0$ over 100 trained nets, after the nets have been trained for 6000 cycles. The dotted line shows the error for the standard A_{R-P}. The solid line shows the error with one QACE and "bounded" internal reinforcement $r^* \in \{0.0, \ldots 1.0\}$.

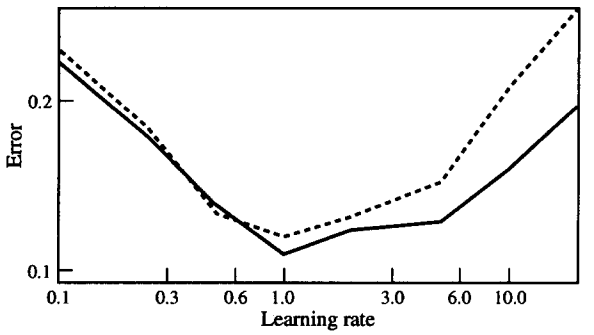

FIGURE 21. Average log error $\bar{e}_0$ versus log α, for sigma–pi based 8-3-8 encoder with eight training vectors having four set-bits. The graph shows the average error $\bar{e}_0$ over 100 trained nets, after the nets have been trained for 6000 cycles. The dotted line shows the error for the standard A_{R-P}. The solid line shows the error with a multi-cube QACE and "bounded" internal reinforcement $r^* \in \{0.0, \ldots, 1.0\}$.

training the 8-3-8 encoder net. If one considers the reduction in average error, $\bar{e}_0$, over all eight learning rates, the overall improvement was 9% better than the standard associative reward-penalty results shown by the dotted line in Figure 18.

8.5. Quantised Adaptive Critic and Associative Reward–Penalty Training

The following figures give the simulation results for one QACE; Figure 19 shows the effect of "bounded" internal reinforcement $r^* \in \{0.0, \ldots, 1.0\}$ and Figure 20 the "unbounded" $r^* \in \{-1.0, \ldots, 2.0\}$. The graphs presented show that the quantised adaptive critic and associative reward–penalty results give lower average errors after 6000 training cycles, for learning rates of greater than $\alpha = 1.0$ for the "bounded" and "unbounded" A_{R-P} regimes. The average error over all eight learning rates was 10% lower for the "bounded" QACE regime than for the standard A_{R-P} training rule, while the average error over all eight learning rates was 11% lower for the "unbounded" QACE regime than for the standard A_{R-P} training rule.

8.6. Multi-Cube Quantised Adaptive Critic and Associative Reward–Penalty Training

Finally the multi-cube QACE is simulated, with four 2-tuples covering the input retina of the 8-3-8 encoder. Figure 21 shows the effect of "bounded" internal reinforcement, $r^* \in \{0.0, \ldots, 1.0\}$, and Figure 22 the "unbounded" internal reinforcement, $r^* \in \{-1.0, \ldots, 2.0\}$.

The two graphs show that the "bounded" multi-cube QACE and the A_{R-P} methodology, while the "unbounded" multi-cube QACE and A_{R-P} training gave a further reduction of the overall average error

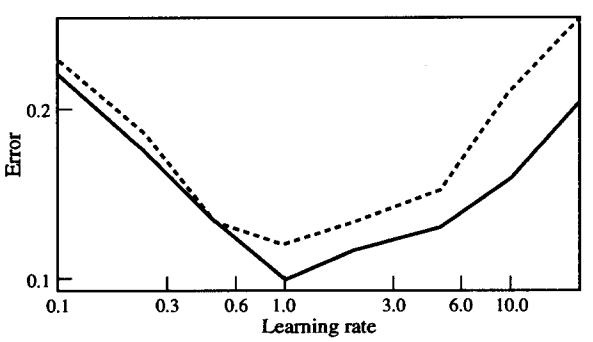

FIGURE 20. Average log error $\bar{e}_0$ versus log α, for sigma–pi based 8-3-8 encoder with eight training vectors having four set-bits. The graph shows the average error $\bar{e}_0$ over 100 trained nets, after the nets have been trained for 6000 cycles. The dotted line shows the error for the standard A_{R-P}. The solid line shows the error with one QACE and "unbounded" internal reinforcement $r^* \in \{-1.0, \ldots, 2.0\}$.

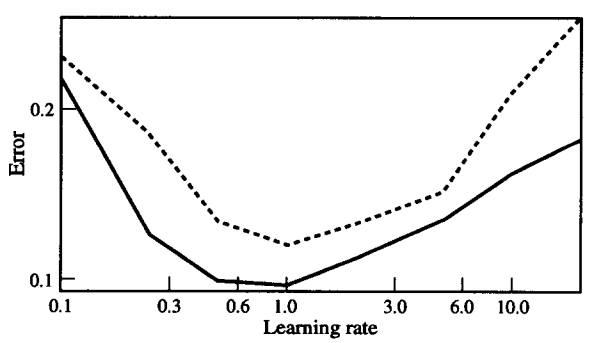

FIGURE 20. Average log error $\bar{e}_0$ versus log α, for sigma–pi based 8-3-8 encoder with eight training vectors having four set-bits. The graph shows the average error $\bar{e}_0$ over 100 trained nets, after the nets have been trained for 6000 cycles. The dotted line shows the error for the standard A_{A-P}. The solid line shows the error with a multi-cube QACE and "ubounded" internal reinforcement $r^* \in \{-1.0, \ldots, 2.0\}$.

of a 19% drop over all eight learning rates when compared with the standard A_{R-P} training regime, the dotted line in the graph.

9. DISCUSSION

Our research into an adaptive critic for sigma–pi networks utilised an unsupervised critic net in parallel with a supervised logical neural network. The critic provides advisory information to the net which was found to augment the net's training efficiency. These techniques use what has been termed an "adaptive critic element" to give critical advice to the sigma–pi network which has been given a specific task. This leads us to the more notable achievement of the presentation of an extension to the associative reward-penalty algorithm, which utilised a quantised adaptive critic element (QACE) and "unbounded" internal reinforcement signal, which permits penalisation of a net even when the penalty coefficient, λ_{rp}, is set to zero. It should be noted, of course, that it is normally the case that the net is only penalised if the penalty coefficient is non-zero, $0 \leqslant \lambda_{rp} < 1$.

It is of interest to note that in the field of (logical) neural networks a limitation may exist which stems from the fact that it is normally the case that supervised learning regimes only rely on an error figure, which only relates to the present input pattern or batch of present input patterns, to adapt the weights of the net while training. This may be a limitation as no other knowledge is utilised during training, while the QACE and the A_{R-P} training methodology utilises data that relates to past and present data (events).

In this research we study the effects of using an unsupervised quantised adaptive critic placed hierarchically above a supervised sigma–pi network, where the critic monitors the input stimuli and traces the frequency of "stimuli" occurrence. This was then used to derive predictions of reinforcement, based on the past and present predictions. The predictions of reinforcement are then used to calculate an internal reward signal.

The investigation into an adaptive critic for sigma–pi structures introduces one to the concept of utilising two networks in parallel, working toward the same goal of optimising a function over time. This work involved carrying over methodologies from the field of semi-linear structures to our sigma–pi units or cube-based logical nodes and the realisation that by utilising the conventions of sigma–pi nodes (e.g., quantisation of variables) one may implement the quantised adaptive critic in hardware.

10. DEVELOPMENT OF IDEAS

Our investigation, which relates to the adaptive learning rule work we previously carried out (Neville & Stonham, 1992, 1994a), studies how to further increase the associative reward–penalty's learning rule efficiency. We used an adaptive critic to obtain more information from the incoming data to better adjust the reinforcement signal to the net.

The research into the adaptive critic evolved as an extension of our initial work with adaptive associative reward–penalty training where we developed adaptive training regimes (Neville & Stonham, 1992, 1994a). Here we looked at the problem from a different perspective, which was to utilise predictive techniques in order to define an "internal" reinforcement. We used the same methodology for the adaptive critic that we utilise for sigma–pi units, which was that each variable was quantised and these were then stored in addressable locations in n-tuples. We also initially limited or "bounded" the value of the "internal" reward signal to the more normal {0, 1} (Barto & Jordan, 1987). The internal reinforcement was derived from predictions of the reinforcement which are in turn derived from eligibility traces. This turned out to be a non-optimal solution and when we stopped restricting the "internal" rewards value, as was the case for the "unbounded" ($r^*_{(t)} \in \{-1.0, \ldots, 2.0\}$), we obtained an extension to the reinforcement training methodology which permits penalisation of a sigma–pi net even when the penalty coefficient, λ_{rp}, was set to zero. Normally the net is only penalised if the penalty coefficient is non-zero, $0 < \lambda_{rp} < 1$. This "unbounded" approach may have repercussions if it is carried over to research on semi-linear units.

The other novel feature of the work on quantised adaptive critics was the realisation that one may utilise the multi-cube structure to enable storage of its variable to be linearly scalable, as the number of inputs increases.

11. CONCLUDING REMARKS

The results for the series of simulations are given in Table 1. Here we compare the three different methods with the standard associative reward-penalty, A_{R-P}, learning rule. The percentage figure represents the

TABLE 1
Average Reduction of Error for the Eight Learning Rates

	"Bounded"	"Unbounded"
Real-value A_{R-P}[9]	9%	—
Single QACE	10%	11%
Multi-cube QACE	10%	19%

average percentage reduction of the average error, $\bar{e}_0$, over all eight learning rates when compared with the standard associative reward–penalty, A_{R-P}, update rule.

When comparing all the quantised adaptive critic elements (QACEs) with real-valued associative reward–penalty ($A^{\Re}_{R-P}$) we see that the quantisation of the prediction and eligibility trace values does not reduce the training efficiency.

The "unbounded" quantised reinforcement QACE gives increased efficiency of training when compared to the "bounded" version. It is of interest that the "unbounded" multi-cube QACE induces more efficient training, this may be explained by noting that the net is given more information while training. The increase of information provided to the net is due to:

(a) The "unbounded" reward signal's ability to reward–penalise the net to a higher degree. If, for example, the external reward provides a penalty signal and the temporal difference between the predictions is a negative quantity (i.e., the $\gamma P_{v(t)} < P_{v(t-1)}$) the prediction of reinforcement is reduced, and then the critic advocates penalisation of the net to a greater degree. The critic may also increase the reinforcement if the external reinforcement signal is a reward and the temporal difference between the predictions is positive (i.e., $\gamma P_{v(t)} > P_{v(t-1)}$).

(b) When utilising a multi-cube [i.e., multiple n-tuples, with an output sigma that sums the presently addressed sites in each tuple to give an average (mean) value (re. Section 7.5)] to store the predictions and trace values one deviates from the original concept of Barto et al. (1983). Barto utilises what he terms "one in a box" coding to provide an input address for the associative search element (ASE) and the adaptive critic element (ACE), which is analogous to a single n-tuple, where each input address only addresses one site. One should, of course, note that the term "n-tuple", which is used as a sigma–pi or logical node stores the site-values in an n-tuple, which is addressed by an n bit input vector $\{x_1, x_2, \ldots, x_n\}$. The methodology of "$n$-tuple" pattern recognition techniques was originally researched by Bledsoe and associates (1959, 1962). In the case of logical or RAM-based nodes, mapped to a digital input retina, each pixel in the input image is represented by a single bit. A number "n" of such bits taken from the input image form an n-tuple input vector. In the case of the single-cube sigma–pi unit the input addresses a single site, μ, which contains a site-value, S_μ, which determines the probability of outputting a logical "1", while in multi-cube structures the output is defined by summing the sites in the sub-cubes and dividing by the number of sub-cubes to obtain a probability of outputting a logical "1". This means, in the case of the multicube QACE, that when one utilises "partial" predictions (e.g., each sub-cube only provides a "partial" prediction, which are then summed to provide the total prediction) one is in effect taking into account information which relates to the occurrence of the "partial" addresses in the input stimuli.

In this research we have put forward the quantised adaptive critic methodology which enables one to extend the original version of A_{R-P} for sigma–pi nets by encapsulating a critic element in the environment to enable the reinforcement signal to provide more detailed information to the net while it is being trained.

We have also overcome one of the problems of the QACE, which is the exponential growth of resources when the number of inputs to it increases, by utilising a multi-cube structure. A notable achievement of this research is the presentation of an extension to the reinforcement training methodology that utilises an "unbounded" reinforcement signal, which permits penalisation of a net even when the penalty coefficient is set to zero, normally the net is only penalised if the penalty coefficient is non-zero.

REFERENCES

Aleksander, I. (1989a). Canonical neural nets based on logic nodes. In *1st IEE International Conference on Artificial Neural Networks* (pp. 110–114).

Aleksander, I. (1989b). The logic of connectionist systems (Chapter 8 pp. 135–155), a probabilistic logic neuron network for associative learning (Chapter 9, pp. 156–171) (see also Chapters 1, 10 and 11). In: I. Aleksander (Ed.), *Neural computing architectures: the design of brain-like machines*, North Oxford Academic Publishers Ltd.

Aleksander, I. (1990). Weightless neural tools: towards cognitive macrostructures. In: *CAIP Neural Network Workshop*. New Jersey: Rutgers University.

Barto, A. (1992). Reinforcement learning and adaptive Critic methods. In *Handbook of intelligent control: neural, fuzzy, and adaptive approaches* (pp. 469–491). New York: Van Nostrand.

Barto, A., & Jordan, M. (1987). Gradient following without back-propagation in layered networks. In *Proceedings 1st IEEE Conference on Neural Networks* (pp. II.629–II.636).

Barto, A., Sutton, R., & Anderson, C. (1983). Neuronlike adaptive elements that can solve difficult learning problems. *IEEE Transactions on Systems Man, and Cybernetics*, **SMC-13**(5), 834–846.

Bledsoe, W. W., & Browning, I. (1959). Pattern recognition and reading machines. In *Proceedings of the Eastern Joint Computer Conference* (pp. 225–232).

Bledsoe, W. W., & Blisson, C. L. (1962). Improved memory matrices for the n-tuple pattern recognition method. *IRE Transactions on Electronic Computers*, **ED11**, 141–415.

Clarkson, T., Gorse, D., Taylor, J., & Ng, C. (1992). Learning

probabilistic RAM nets using VLSI structures. *IEEE Transactions on Computers*, **41**, 1552–1561.

Gorse, D., & Taylor, J. (1990a). Reinforcement training strategies for probabilistic RAMs. In *Theoretical aspects of neurocomputing: selected papers from the symposium on neural networks and neurocomputing* (pp. 180–184). NEURONET90.

Gorse, D., & Taylor, J. (1990b). Training strategies for probabilistic RAMs. In *Parallel processing in neural systems and computers* (pp. 161–164). Amsterdam: North-Holland.

Gorse, D., & Taylor, J. (1991). A continuous input RAM-based stochastic neural model. *Neural Networks*, **4**, 657–665.

Gullapalli, V. (1988). *A stochastic algorithm for learning real-valued functions via reinforcement feedback*. COINS Technical Report Number: 88-91, September 29th. The Department of Computer and Information Sciences (COINS), The University of Massachusetts.

Gurney, K. (1989). *Learning in nets of structure hypercubes*. PhD thesis, Department of Electrical Engineering, Brunel University, Manchester (available as: Technical Memo CN/R/144).

Gurney, K. (1992a). Training nets of hardware realisable sigma–pi units. *Neural Networks*, **5**, 289–303.

Gurney, K. (1992b). Weighted nodes and RAM-nets: A unified approach. *Journal of Intelligent Systems*, **2**(1–4), 155–185.

Gurney, K. (1993). Training nets of stochastic units using system identification. *Neural Networks*, F3 > 6(1), 133–145.

Hinton, G., Sejnowski, T., & Ackley, D. (1984). *Boltzmann machines: constraint satisfaction networks that learn*. Technical Report CMU-CS-84-119, Carnegie Mellon University, Pittsburgh, PA.

Hinton, G., Ackley, D., & Sejnowski, T. (1985). A learning algorithm for Boltzmann machines. *Cognitive Science*, **9**, 147–169.

Hui, T., Bolouri, P., & Gurney, K. (1992a). VLSI implementation of digital neural network with reward-penalty learning. *3rd International Workshop on VLSI for Neural Networks and Artificial Intelligence*, Oxford University.

Hui, T., Gurney, K., & Bolouri, P. (1992b). A cascadable 2048-neuron VLSI artificial neural network with on-board learning. *International Conference on Artificial Neural Networks*, Brighton, UK.

Kan, W.-K., & Aleksander, I. (1987). A probabilistic logic neuron network for associative learning. *Proceedings IEEE International Conference on Neural Networks*, San Diego, Volume II, pp. II.541–II.548.

Minsky. M. (1961). Steps towards artificial intelligence. *IRE*, **49**, 8–30.

Minskey, M., & Papert, S. (1969). *Perceptrons: an introduction to computational geometry*. Cambridge, MA: The MIT Press.

Myers, C. (1989). Output functions for probabilistic logic nodes. *1st IEE International Conference on Artificial Neural Networks*.

Myers, C. (1990). *Learning with delayed reinforcement in an exploratory probabilistic logic neural network*. PhD thesis, Imperial College of Science, The University of London, Department of Electrical Engineering.

Myers, C. & Aleksander, I. (1988). Learning Algorithms for Probabilistic Neural Nets, 1st International Neural Network Society Annual Meeting, Boston, USA.

Neville, R. (1990). *Investigate and evaluate the design of probabilistic nodes for boolean n-cube networks*. Master's thesis, Department of Electrical Engineering, Brunel University.

Neville, R. (1993). *Augmentation of sigma–pi structures and learning regimes*. PhD thesis, Department of Electrical Engineering, Brunel University, Middlesex.

Neville, R., & Stonham, T. (1992). Adaptive reward–penalty for probabilistic logic nodes. *International Conference on Artificial Neural Networks*, IJCNN-92, Brighton.

Neville, R., & Stonham, T. (1993). Adaptive critic for probabilistic logic nets. *World Congress on Neural Networks* (pp. III.389–III.392), Portland, Oregon.

Neville, R., & Stonham, T. (1994a). Adaptive associative reward–penalty algorithms for sigma–pi networks. *International Journal of Neural, Parallel and Scientific Computations*, June, **2**(2), 141–164.

Neville, R., & Stonham, T. (1994b). Unbounded reinforcement for the associative reward-penalty algorithm. *The World Congress on Neural Networks*, WCNN-94 (pp. III.637–III.640), San Diego, CA.

Neville, R., & Stonham, T. (1994c). A comparison study of unbounded and real-valued reinforcement associative reward-penalty algorithms. *The International Conference on Artificial Neural Networks*, ICANN-94 (pp. 651–654), Sorrento, Italy.

Neville, R., & Stonham, J. (1995). Generalisation in sigma–pi networks. *Connection Science: Journal of Neural Computing, Artificial Intelligence and Cognitive Research*, March, **7**(i) 29–59.

Neville, R., Glover, R. J., & Stonham, J. (1995a). Evaluation of training sigma–pi networks on a massively parallel processor. World Congress on Neural Networks, WCNN-95, Washington, DC (pp. II-484–II-487). 1995, International Nueral Network Society Annual Meeting, Renaissance Hotel Washington, DC July 17–21, Volume II, ISBN 0-8058-2125-2.

Neville, R., Glover, R. J., & Stonham, J. (1995b). Evaluation of training & mapping sigma–pi networks to a massively parallel processor. *IEEE ICNN'95, International Conference on Neural Networks*, Perth, Western Australia, 27 November–1 December.

Neville, R., Glover, R. J., & Stonham, J. (1995c). *Mapping neural networks to the associative string processor*. Department of Electrical Engineering, Brunel University, England, Registered as Technical Memorandum N/R/150 in the Department of Electrical Engineering, Brunel University, Copyright © October 1995.

Penny, W., Gurney, K., & Stonham, T. (1990). Reward-penalty training for logical neural networks. In *International Conference on Artificial Intelligence, Applications and Neural Networks* (pp. 26–29), Zurich.

Rumelhart, D., McClelland, J., and the PDP Research Group (1986). *Parallel distributed processing*. Cambridge, MA: MIT Press.

Thorndike, E. (1911). *Animal intelligence*. Darien, CT: Hafner.

Werbos, P. (1989). Back-propagation and neurocontrol: A review and prospectus. *IEEE Proceedings of the International Joint Conference on Neural Networks* (IJCNN'89) (pp. 1:I.209–I.216), New York.

Werbos, P. (1990). Consistency of HDP applied to a simple reinforcement learning problem. *Neural Networks*, **3**, 179–189.

Werbos, P. (1992a). Approximate dynamic programming for real-time control and neural modelling. In *Handbook of intelligent control: neural, fuzzy and adaptive approaches* (pp. 493–525), New York: Van Nostrand.

Werbos, P. (1992b). Neurocontrol and supervised learning: An overview and evaluation. In *Handbook of intelligent control: neural, fuzzy and adaptive approaches* (pp. 65–89), New York: Van Nostrand.

Werbos, P. (1992c). Neurocontrol and fuzzy logic: Connections and design. *International Journal of Approximate Reasoning*, **6**, 185–219.

Widrow, B., & Hoff, M. (1960). Adaptive switching circuits. *1960 IRE Western Electric Show and Convention Record*, Part, 4, 96–104.

Wilkie, B. A. (1983). *A stand-alone, high resolution, adaptive pattern recognition system*. PhD Thesis W499, Department of Electrical Engineering, Brunel University, Middlesex.

Williams, R. (1986). *Reinforcement learning in connectionist networks: a mathematical analysis*. Technical Report ICS Report 8605, Cognitive Science, University of California, San Diego, CA.

Williams, R. (1987a). A class of gradient-estimation algorithms for reinforcement learning in neural networks. *Proceedings of the International Conference on Neural Networks* (Vol. II; pp. II.601–II.608), San Diego, CA.

Williams, R. (1987b). *Reinforcement-learning connectionist systems.* Technical Report NU-CCS-87-3, College of Computer Science, Northeastern University, Boston, MA.

Witten, I. H (1977). An adaptive optimal controller for discrete-time Markov environments. Inform. Contr., **34**, 286–295.

NOMENCLATURE

a_i	real-valued activation of the ith node
w_{ij}	real-valued weight between node i and j
$\sigma(\cdot)$	sigmoid function
$\sigma'(\cdot)$	reads as "sigmoid prime", represents the derived function of $(\cdot)$
μ	hypercube site or addresses μ
S_μ	site-value at site-address
$P(X)$	probability of event X occurring
x	binary input vector, which may be represented by a set of bits $\{x_1, x_2, \ldots, x_n\}$
y	binary output
$\langle k \rangle$	expectation of variable k
k^*	estimate of value of k
$\{q\|s\}$	set of elements of q for which the property s holds
r	binary scalar reward signal
Y_t^i	binary target output response
$\Delta(v)$	incremental (or δ) change in value of v
α	learning rate
λ	punishment coefficient; sets ratio of punishment to reward in the training regime
ρ	parameter governing steepness of sigmoidal function
$sgn(\Delta S_\mu)$	is the sign of the delta change ΔS_μ (i.e., if $\Delta S_\mu = 1.25$ then $sgn(\Delta S_\mu)u = +1$ and if $\Delta S_\mu = -1.6$ then $sgn(\Delta S_\mu) = -1$)