Stand-Alone Hardware-Based Learning System

Trevor CLARKSON and Chi Kwong NG[1]

Department of Electronic and Electrical Engineering,

King's College, Strand, London, WC2R 2LS, UK

[1]Department of Electronic Engineering, City Polytechnic of Hong Kong, Hong Kong

(Received

The probabilistic Random Access Memory (pRAM) is a biologically-inspired model of a neuron. The pRAM behaviour is described in this paper in relation to binary and real-valued input vectors. The pRAM is hardware-realisable, as is its reinforcement training algorithm. The pRAM model may be applied to a wide range of artificial neural network applications, many of which are classification tasks. The application presented here is a control problem where an inverted pendulum, mounted on a cart, is to be balanced. The solution to this problem using the pRAM-256, a VLSI pRAM controller, is shown.

KEYWORDS: neural networks, VLSI, reinforcement training, pRAM

17. Introduction

The probabilistic RAM[1, 2] (pRAM) device has been recently described[3] as an example of VLSI implementation of an artificial neural network. The pRAM neuron[2, 4] generates an output in the form of a spike train where the probability of generating a spike is controlled by an internal weight, represented as a real-valued number. The firing probabilities for all possible binary input vectors can be trained and 2^N weights are used in each pRAM, where N is the number of synaptic inputs to the pRAM.

The pRAM is a hardware device with intrinsically neuronlike behaviour (Fig. 1). It maps binary inputs (representing the presence or absence of a pulse on each of N input lines) to a binary output (1 being equivalent to a firing event; 0 to inactivity). This mapping from $\{0,1\}^N$ to $\{0,1\}$ is, in general, a stochastic function. If the address locations in an N-input pRAM are indexed by an N-bit binary address vector **i** using an address decoder, the output a is 1 with probability

$$\text{Prob}\,(a = 1 \mid \mathbf{i}) = \alpha_i;, \tag{1}$$

where $\mathbf{i} \in \{0,1\}^N$ is the vector representing input activity. The quantity α_i represents the firing probability for the binary input vector **i**. In the hardware realisation of the device α_i is represented as an M-bit integer for each memory location having a value in the range of 0 to $2^M - 1$, and these values represent probabilities in the range $\{0, \frac{1}{2^M}, \frac{2}{2^M}, ..., 1 - \frac{1}{2^M}\}$. α_i may be assigned values which have a neurobiological interpretation; it is this feature which allows networks of pRAMs, with suitably chosen memory contents, to closely mimic the behaviour of living neural systems.

In a pRAM, all 2^N memory components are independent random variables. Thus, in addition to possessing a maximal degree of nonlinearity in its response function - a

deterministic, $\alpha \in \{0,1\}^N$, pRAM can realise any of the 2^{2^N} possible binary functions of its inputs - pRAMs differ from units more conventionally used in neural network applications in that noise is introduced at the synaptic rather than the threshold level; it is well known that synaptic noise is the dominant source of stochastic behaviour in biological neurons.

The pRAM models the noise which arises from the release of neurotransmitter by single vesicles in the synapses of real neurons. Each vesicle releases a variable amount of neurotransmitter which may cause the neuron to fire with a given probability. This feature is represented by the α_i in the pRAM, which is the firing probability for a given input vector, **i**, at the neuron's synapses at that instant. An *N*-input pRAM has 2^N weights which represent the firing probabilities for all possible combinations of input. Thus $\alpha_{00\cdots0}$ is the firing probability when there is no input activity (spontaneous firing) and $\alpha_{00\cdots0}$ is the firing probability when there is activity only on the first synapse, and so on.

A typical Perceptron model[5)] of a neuron possessing *N* inputs has *N* weights. For such a model, multiple vesicle releases are modelled such that, at each synapse, the mean neurotransmitter release per vesicle is called the weight. For each synapse, this weight is multiplied by the input synaptic activity and these products are summed before thresholding, or a squashing function, is applied to determine the neuron's output. It can be seen that nonlinear interactions between synapses cannot be represented in a single neuron of this form, whereas the pRAM can exhibit nonlinear behaviour in terms of synaptic activity. Linear-weighted-sum neurons can be made noisy, but this is normally achieved by superimposing noise at the point of summation or at the threshold level.

It is also noted that because pRAM networks operate in terms of 'spike trains' (streams of binary digits produced by the addressing of successive memory locations), information concerning the timing of firing events is retained; this potentially allows phenomena such as

the observed phase-locking of visual cortical neurons to be reproduced by pRAM nets, with the possibility of using such nets as part of an effective 'vision machine'.

18. Reinforcement Training

Reinforcement training is a strategy used in problems of adaptive control in which individual behavioural units (of which pRAMs are an example) only receive information concerning the quality of the performance of the system as a whole, and must discover for themselves how to change their behaviour so as to improve this performance. Because it relies only on a global success/failure signal, reinforcement training is likely to be the method of choice for 'on-line' neural network applications.

A form of reinforcement training for pRAMs has been devised which is fast and efficient. Networks of such units are likely to find wide application, for example, in the control of autonomous robots. Control need not be centralised; small nets of learning pRAMs could, for example, be located in the individual joints of a robot limb. Such a control arrangement would, in many ways, be akin to the semi-autonomous neural ganglia found in insects.

The form of the training rule devised for the pRAM[6] is described by

$$\Delta\alpha_{\mathbf{u}}(t) = \rho((a - \alpha_{\mathbf{u}})r + \lambda(\bar{a} - \alpha_{\mathbf{u}})p)(t) \times \delta_{\mathbf{u},\mathbf{i}};, \tag{2}$$

where r and p are global success or failure signals respectively, $\in \{0, 1\}$, received from the environment at time t, $a(t)$ is the unit's binary output, $\alpha_{\mathbf{u}}$ is the weight addressed by the binary input vector $\mathbf{u}$, and ρ and λ are constants $\in [0,1]$. The delta function is included to make it clear that only the location which is actually addressed at time t is available for modification, the contents of the other locations being unconnected with the behaviour that led to reward or punishment at time t. When r = 1 (success) the probability $\alpha_{\mathbf{u}}$ changes so as to increase the chance of emitting the same value from that location in the future, whereas if p = 1 (failure) the probability of emitting the other value when addressed increases. The constant λ

represents the ratio of punishment to reward; a nonzero value for λ ensures that training converges to an appropriate set of memory contents and that the system does not become trapped in false minima. Note that reward and penalty take effect independently; this allows the possibility of 'neutral' actions which are neither punished nor rewarded but may correspond to a useful exploration of the environment.

Figure 2 shows the manner in which rule (2) has been implemented in hardware. The memory contents $\alpha_{\mathbf{u}}(t+1)$ are updated each clock period according to rule (2).

There are many interesting problems of adaptive control which require real-valued inputs. These inputs, when digitised, may be input to a pRAM network in parallel as shown later in the paper. It is also possible to input such a real-valued vector in the form of a spike train to a single pRAM input. The pRAM is modified to include spike generators for the inputs (pRAMs) and spike integrators at the outputs (counters) which will enable such inputs or outputs to be handled. Such a modified device is called an integrating pRAM.

19. Integrating pRAM

This device performs mappings from $[0,1]^N$ to $\{0,1\}$ using the concept of time-averaging. The integrating pRAM, or i-pRAM, is shown in Fig. 3. A real-valued input vector $\mathbf{x} \in [0,1]^N$ is approximated by the time-average (over some period R) of successive binary input patterns $i \in \{0,1\}^N$ by the real to spike-frequency translator which is normally a pRAM.

$$x_{\mathbf{u}} = \frac{1}{\mathrm{R}} \sum_{\mathrm{r}=1}^{\mathrm{R}} i_{\mathbf{u}}(r) \qquad (3$$

At each time step r = 1...R=2^M, $\mathbf{i}(r)$ selects a particular location in the pRAM using the address inputs which result in a binary output $\hat{a}(r)$. These outputs are accumulated in a spike integrator (Figure 3) whose contents are reset at the start of a cycle. After R time steps, the

contents of the counter are used to generate the binary i-pRAM output (4)

which is 1 with probability

$$\text{Prob}(a = 1 \mid \mathbf{x}) = \frac{1}{R}\sum_{r=1}^{R} \hat{a}\,(r)$$

$$= \sum_{\mathbf{u}} \alpha_{\mathbf{u}} \prod_{j=1}^{N} (x_j u_j + \bar{x}_j \bar{u}_j);.$$

This i-pRAM can be developed further to implement a generalised form of the training rule (2). According to rule (2), the input of a single binary address results in the contents of the single addressed location being modified. However, the i-pRAM can be used to implement a generalised form of training rule (2) in which the input of a real-valued number causes the contents of multiple locations to be modified.

Address counters are required for counting the number of times each of the storage locations is addressed. This device can then be used to implement the generalised training rule referred to above. This generalised training rule[7] is

$$\Delta\alpha_{\mathbf{u}}(t) = \rho((a - \alpha_{\mathbf{u}})r + \lambda(\bar{a} - \alpha_{\mathbf{u}})p)(t)X_{\mathbf{u}}(t) \quad , \qquad 5)$$

where $X_{\mathbf{u}(t)}$ replaces the delta function in eq. (2). Thus in the learning i-pRAM case, every location is available for updating, with the change being proportional to that address's responsibility for the ultimate i-pRAM binary output a(t). The $X_{\mathbf{u}}$'s record the frequency with which addresses have been accessed and are derived from the address counters above.

The learning rule may be further generalised in order to deal with situations in which reward or punishment may occur an indefinite number of time steps after the critical action which caused the environmental response.[7] In such delayed-reinforcement tasks it is necessary to learn path-action rather than position-action associations. This can be done by adding *eligibility traces* to each memory location. These decay exponentially, by a factor δ, where a location is not accessed, but otherwise are incremented to reflect both access frequency and the resulting i-pRAM action. One trace records "access and activity", whereas

a complementary trace records "access and inactivity" (both are equally important in developing an appropriate response to a changing environment). These terms are called $e_{\mathbf{u}}(t)$ and $f_{\mathbf{u}}(t)$, respectively. The eligibility traces are initialised to zero at the start of a task and subsequently updated according to

$$e_{\mathbf{u}}(t) = \delta e_{\mathbf{u}}(t-1) + \bar{\delta}\mathrm{a}(t)\mathrm{X}_{\mathbf{u}}(t) \tag{6}$$

$$f_{\mathbf{u}}(t) = \delta f_{\mathbf{u}}(t-1) + \bar{\delta}\bar{\mathrm{a}}(t)\mathrm{X}_{\mathbf{u}}(t)\,. \tag{7}$$

The necessary extension of eq.(5), which results in the capacity to learn about temporal features of the environment, is

$$\Delta\alpha_u(t) = \rho((\overline{\alpha_{\mathbf{u}}}e_{\mathbf{u}} - \alpha_{\mathbf{u}}f_u)\mathrm{r} + \lambda(\overline{\alpha_{\mathbf{u}}}f_{\mathbf{u}} - \alpha_{\mathbf{u}}e_{\mathbf{u}})\mathrm{p})(t)\,. \tag{8}$$

When $\delta = 0$, $e_{\mathbf{u}} = \mathrm{a}\,\mathrm{X}_{\mathbf{u}}$, and $f_{\mathbf{u}} = \bar{\mathrm{a}}\,\mathrm{X}_{\mathbf{u}}$. It can be seen that in this case eq.(8) reduces to the original learning i-pRAM training rule (5).

20. VLSI pRAMs

The pRAM-256 is a VLSI device which processes 256 internal pRAMs in a similar manner to an earlier pRAM device[3] (Fig. 4). The earlier device[3] provided only local reinforcement training in which two "auxiliary pRAMs" are used to determine the appropriate reward and penalty signals for each "learning pRAM". The pRAM-256 device is no longer restricted to local learning. By allowing the reward and penalty inputs to each pRAM to be reconfigured by the use of "connection pointers", global, local and competitive methods of training can be used in this fourth generation of pRAM hardware.[8]

The configuration of the pRAM network is defined by a "connection pointer" table where one entry exists for each input of each pRAM. This table is held in static RAM (SRAM) with the pRAM weight memory (Fig. 4). Thus many architectures can be built with the same hardware; reconfiguring the network only requires updating a connectivity table.

Processing of neurons in the pRAM-256 is carried out in two passes. The first pass provides forward processing of signals through the network and updates the outputs of each of the 256 pRAMs. The second pass is only executed if training is enabled, and updates the pRAM weights used in pass 1 in response to the new output of the net. Pass 1 processing requires 154μs and processing both Pass 1 and Pass 2 requires 246μs in total, both times are for a clock of 33MHz.

At the same time as the pRAMs are being processed, new external data may be shifted onto the chip so that neural processing may proceed without incurring any data transfer delay.

21. Hardware Learning

The learning algorithm used in the pRAM-256 is given by eq.(2) above. Thus it may be seen from eq.(2) that the weight, α_u, is always in the interval [0,1] so that clipping is never required. The action of the learning algorithm is to move the weight closer to the value of *a* (the pRAM output) if a reward is given and to move the weight further away from *a* if a penalty signal is given, for binary inputs and outputs. Thus beneficial actions are made more likely to happen in the future and adverse actions are made less likely to occur, given the same circumstances. It is the use of a hardware-realisable algorithm on-chip which makes a totally hardware-based learning system possible. This algorithm is incorporated within the pRAM-256 chip shown in Fig. 4.

22. Interfacing to External Hardware

To interface the pRAM-256 device to external hardware, programmable logic devices (FPGAs) are currently used. These FPGAs have many registers, some of which may be used to count the spikes coming from the serial outputs of individual pRAMs to generate a mean firing frequency. Combinational circuitry within the FPGA may be used for the *environment*, to generate the reward and penalty signals during training.

An example of the use of the pRAM-256 to solve a pattern classification task has been given by Clarkson and Ng[9]. The next section shows how a pRAM network may be used to solve a classical pole-balancing task.

23. Inverted Pendulum System

The inverted pendulum system is a classical problem that can be used to test the performance of a neural network.[9] In this experiment, the system comprises a rigid pole and a cart on which the pole is hinged. In order to reduce the complexity of the problem, the cart moves on rails to its right or left, depending on the force exerted on the cart. The pole is hinged to the cart through a frictionless free joint which allows the pole to move in one dimension only. Figure 5 shows the arrangement of the pendulum system. The objective of the neural network control system is to balance the pole starting from nonzero conditions by applying appropriate force to the cart.

The inverted pendulum system of Fig. 5 can be characterised by the second-order differential equations[11]

$$\ddot{\theta} = \frac{g \sin\theta + \cos\theta \left(\frac{-F - m\, l\, \dot{\theta}^2 \sin\theta}{m_p + m} \right)}{l \left(\frac{4}{3} - \frac{m \cos^2\theta}{m_p + m} \right)}$$

$$\ddot{x} = \frac{F + ml(\dot{\theta}^2 \sin\theta - \ddot{\theta} \cos\theta)}{m_p + m},$$

where the state variables are

θ angle of the pole with respect to the vertical axis,

$\dot{\theta}$ angular velocity of the pole,

x position of the cart on the track,

$\dot{x}$ linear velocity of the cart, and

F is the applied force in newtons,

and the values of the constants are

g (acceleration due to gravity) = 9.8 ms^{-2},

m (mass of the cart) = 1.0 kg,

m_p (mass of the pole) = 0.1 kg,

l (half-length of the pole) = 0.5 m.

The objective of this control system is to balance the pole without controlling the absolute position of the cart; therefore only the first equation is relevant to this experiment. The experiment may later be extended for the case where the horizontal position of the cart is controlled within prescribed limits.

23.1Network architecture

In order to generate the appropriate force, *F*, for balancing the pole, the network must be able to estimate the next state variables at time *t+1* provided that the state variables at the present time, *t*, are given. This is done by using an *estimation subnet* which takes $\theta(t), \dot{\theta}(t)$ and *F* as inputs (Fig. 6). The outputs of the estimation subnet are $\theta(t+1)$ and $\dot{\theta}(t+1)$. The subnet is trained to perform the calculation according to the following differential equations:

$$\theta(t+\delta) = \delta\,\dot{\theta}(t) + \theta(t)$$

$$\dot{\theta}(t+\delta) = \delta\,\ddot{\theta}(t) + \dot{\theta}(t)$$

which are the linear approximations of the next-time-step state variables.

The state variables are represented by 6-bit signed binary numbers and F is represented by one bit. The control action F is calculated by another subnet, the *action subnet*, whose structure is shown in Fig. 7. The inputs to this action subnet are the state variables corresponding to the present time step.

The estimation and action subnets are connected to form a *slice network*. The structure of the slice network is depicted in Fig. 4. The action subnet takes inputs from the previous

slice network and produces the corresponding control action (F). The control action and the previous state variables will be fed into the estimation subnet which prepares the prediction of the next-time-step state variables for the adjacent slice network. Therefore, by connecting N units of the slice network, one can predict the movement of the inverted pendulum system and the control goal can be achieved.

The delay introduced by the slice network corresponds to one time step (δ) between two consecutive state variable pairs. In this experiment, the network is used to predict the movement (the angle and angular speed) of the pendulum system from t = 0 to t = 1 s. Obviously, a shorter value for δ requires a larger network because more slice networks are required to cover the time span. Therefore, a large value of δ would be preferred based on this reason. On the other hand, the delay cannot be too long, for if the control action is too slow, the network will no longer be able to balance the pole. In addition, with a long delay, the linear approximation from the estimation subnet may no longer be precise enough and a higher-order approximation, e.g., a second-order approximation, would have to be used instead. This will increase the training time. Since δ is related to the speed of the system clock, we can manipulate the time delay by adjusting the system clock. In this experiment, the pRAM system clock is 33 MHz and the delay induced by a combined forward and weight-update process is 174 ns. For each slice network, 64 samples are taken and the average is sent to the decision maker (Fig. 9). This contributes to a total delay time of 11 ms. In order to cover a time span of 1 second, a network with 90 slice networks is required. The schematic diagram of the complete *p*RAM-based control system is shown in Fig. 9.

The decision maker is an external circuit which performs the following tasks:

- averages the outputs of the slice networks,

- for each slice network, it computes the hamming distance between the estimated pole movement and the desired one,
- it generates the environmental signals (r and p). The calculated hamming distance will be thresholded. If it is lower than a predefined threshold, the corresponding slice network will be rewarded. Otherwise, the slice network will be penalised.

The decision maker uses a 12-bit accumulator to calculate the average of the slice network outputs. After 64 samples have been added to the accumulator, the top 6 bits are taken as the average of the samples.

23.2Training strategy

The network is trained by the on-chip reinforcement training algorithm which is provided in the *p*RAM-256 modules. Every slice network receives a pair of environmental signals (r and p) from the decision maker, hence the learning is localised.

7.2.1 Estimation subnet

Since every slice network uses the same estimation subnet, it is required to train only one estimation subnet. The trained estimation subnet will be replicated accordingly. The training set is generated by solving eq.(9). In order to improve the network's generalisation from the training set, a random noise signal is added during training whose amplitude is up to 10% of the maximum input vector. For shorter training time, the training noise level can be reduced at the expense of generalisation. The elements of the training set are *initial pole angle*, *angular velocity* and *F* (the applied force) together with the *desired pole angle* and *angular velocity*.

All of the above variables are represented using 6-bit signed binary numbers. When there is no training noise, the estimation subnet is trained to perform a one-to-one mapping between present and next-state angles and velocities. Such a one-to-one mapping together

with the stochastic property of the *p*RAM indicates that the estimation subnet is not able to produce a pair of stable outputs. By introducing training noise, the subnet is forced to perform a many-to-one mapping . As a result, the outputs of the subnet will be the same even though there is a slight change in the inputs; hence, a pair of more stable outputs will result.

7.2.2 Action subnet (slice network)

Outputs of the slice networks can be regarded as the trajectory of the pole; every slice network corresponds to a particular point of the pole at a given time. Given the pair of state variables, the action subnet is expected to generate the correct control action which is used to estimate the state variables at the next time step. Whether the control action is good or not depends on the generated state variables. During the training process, the generated state variables will be compared with a pair of target values. If the difference is within a predefined limit, the action subnet will be rewarded, otherwise it will be penalised.

The training set contains the initial conditions and the desired pairs of state variables. The initial condition is a two-dimensional vector which describes the initial position and velocity of the pole. The desired pairs of state variables contain the value of the state variables at different time steps. Since there are 90 slice networks which correspond to 90 time steps, 90 pairs of desired state variables are required.

It is required to halt training when the average of the slice network output is being calculated. The decision maker can do so by setting r=0 and p=0. An alternative strategy is to disable the TRAIN input to the pRAM 256, which results in only forward passes being processed. By means of the strategy above, a pRAM net was trained to solve eq.(9) and thereby to balance the inverted pendulum.

24. Conclusions

It has been shown how the pRAM artificial neuron may process temporal data in two ways. The first is by the use of the integrating pRAM which retains temporal information in the form of an activity history. The second is by dividing time into slices, and this method is closer to pattern recognition than is the first method. We have proposed an application for a neuroprocessor, the *p*RAM-256, to solve the inverted pendulum balancing problem using the second method. This application shows the flexibility of using *p*RAM-256 chips in *p*RAM-based neural network construction. In the system described, ten *p*RAM-256 chips are used. These are connected via the on-chip serial links. In addition, the on-chip reinforcement learning facility enables fast learning for a large-scale network. However, the decision maker is a drawback in a self-contained system. We are investigating the feasibility of incorporating such a facility into a single package.

References

1) D. Gorse and J.G. Taylor: Phys. Lett. A131 (1989) 326.

2) D. Gorse and J.G. Taylor: Physica D (1989) 90.

3) T.G. Clarkson, C.K. Ng and Y. Guan: IEEE Trans. Neural Networks 4 (1993) 408.

4) T.G. Clarkson, D. Gorse, J.G. Taylor and C.K. Ng: IEEE Trans. Comput. 41 (1992) 1552.

5) S. Haykin: Neural Networks, (Macmillan, New York, 1994), Chap. 1, p. 8.

5) D. Gorse and J.G. Taylor: Proc. INNC-90, Paris (Dordrecht, Kluwer, 1990) p. 821.

6) D. Gorse, J.G. Taylor and T.G. Clarkson, Proc. ICONIP'94, Seoul, 1994 p. 293.

7) T.G. Clarkson and C.K. Ng: Proc. Microelectronics for Neural Networks, Edinburgh, Scotland, 1993 (UnivEd, Edinburgh, 1993), p. 233.

8) T.G. Clarkson and C.K. Ng: Proc. Int. Conf. Solid State Devices and Materials, Yokohama, 1994 (Business Center for Academic Societies Japan, Tokyo, 1994) p. 352.

9) A.G. Barto, R.S. Sutton and C.W. Anderson: IEEE Trans. Syst. Man & Cybern., SMC-13 (1983) 834.

10) R.H. Cannon: Dynamics of Physical Systems (McGraw-Hill, New York, 1967).

Fig. 1. The pRAM model.

Fig. 2. The on-chip learning unit which implements eq.(2).

Fig. 3. The integrating pRAM.

Fig. 4. The pRAM-256 neuroprocessor architecture.

Fig. 5. The inverted pendulum system.

Fig. 6. The estimation subnet.

Fig. 7. The action subnet structure.

Fig. 8. A slice network is constructed by connecting an action subnet and an estimation subnet.

Fig. 9. The complete pRAM-based control system for the inverted pendulum.