

# An Empirical Study of Two Approaches to Automated pRAM Network Design

Kwok Ching Tsui and Dr. Mark Plumbley

Department of Computer Science  
King's College London  
Strand, London WC2R 2LS, UK  
fax: +44 171 873 2851  
email: ktsui@dcs.kcl.ac.uk and M.Plumbley@kcl.ac.uk

## Abstract

The application of Genetic Algorithms (GAs) to the automated design of artificial neural networks has received much attention in recent years. A number of common network models have been studied. This paper presents empirical results on the application of GAs to the Probabilistic RAM (pRAM) network model. Pattern recognition tasks are presented to the pRAM networks. These results are compared with those using Simulated Annealing (SA) as well as the reinforcement learning algorithm originally proposed for pRAM networks. We found that both SA and GA do not perform as well as the reinforcement learning algorithm. Nevertheless, GA is able to achieve an average recognition rate of 83.78% at the expense of long running time.

## 1 Introduction

There is increasing interest in the application of machine learning algorithms to assist with design and training of neural networks. One common technique is the use of Genetic Algorithms (GAs) [7], in which a population of individuals in a parallel search for a goal solution. Related techniques include Simulated Annealing [9].

The pRAM [1] is a stochastic artificial neuron with similarities with biological neurons. Although pRAM networks are not as widely studied as the other common network models, various applications of them have been widely published [2, 3, 6]. A significant advantage of pRAM networks over some other neural networks is their ease of implementation in hardware. Learning in a pRAM network, like the other network models, is achieved by performing an *iterated* training process using a learning rule. In particular, a reinforcement learning algorithm has been proposed that can be implemented in hardware [2]. A hardware chip module with 256 pRAMs has been designed and fabricated. The latest version can communicate with 4 other modules, and allows a network with more than 1000 neurons to be built.

The main objective of this study is to gain insight into the applicability and effectiveness of GAs to pRAM network training. For comparison purpose, both simulated annealing and the original reinforcement algorithm are also used.

Section 2 of this paper gives an introduction to the characteristics of pRAM network models with emphasis on the learning algorithm. Pattern recognition tasks are used in the experiments and a brief description can be found in section 3. The following two sections give the details of the simulated annealing and GA techniques studied. Simulation results are given in section 7. We conclude with a discussion of the usefulness of the two algorithms in the design of pRAM networks.

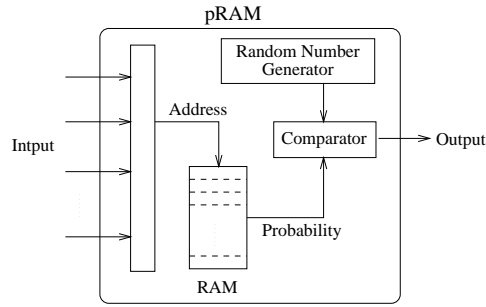


Figure 1: Block diagram of a pRAM.

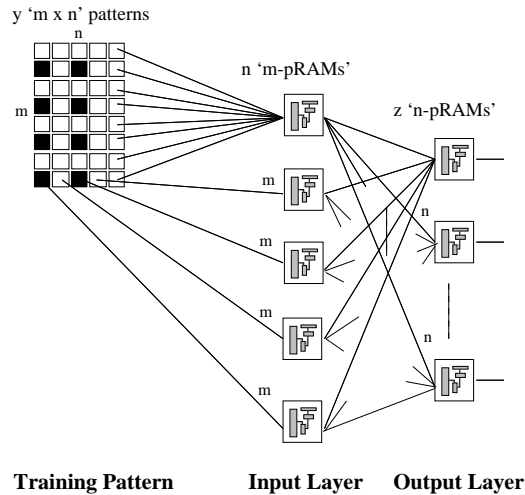


Figure 2: Typical setup for using a pRAM network. For  $y$  input patterns of  $m \times n$  matrix, the number of pRAM at the input layer is equal to the number of columns  $n$  at the input matrix and the type of pRAM depends on the number of rows  $m$ . The number of pRAM at the output layer is equal to the  $z$  where  $2^z$  is equal to the number of input patterns to be presented and the type of pRAM depends on  $n$ .

## 2 Learning in pRAM networks

A pRAM is a biologically-inspired neuron model that exhibits nonlinear stochastic behaviour similar to a biological neuron [2]. The output of the pRAM is a stochastic spike train signal of ‘1’s and ‘0’s.

The pRAM (figure 1) consists of a number of memory locations  $\underline{u}$  that each contain a firing probability  $\alpha_{\underline{u}}$ . An  $n$ -bit binary vector input to the pRAM is decoded as the address of one of the  $2^n$  memory locations. To determine if the pRAM will “fire”, the probability is retrieved, and it is added to a random number of the range  $\{0,1\}$ . If the sum is bigger than 1, the output  $a$  of the pRAM will be ‘1’, or said to have “fired”: otherwise,  $a$  will be ‘0’. Because of the stochastic nature, if a real number is needed the output of a pRAM is averaged for a number of cycles (say, 500) using the same input pattern. The resultant real number is similar to a probability of that pRAM neuron firing in reaction to the input pattern.

A pRAM unit with  $n$  input lines is called an  $n$ -input pRAM and its total number of memory locations is  $2^n$ . A pyramidal pRAM network (figure 2) consists of a number of pRAM units connected in a fashion similar to a feedforward net. A pyramidal pRAM network aligns pRAM units into layers and connections are made between pRAMs in different layers.

A reinforcement learning algorithm has been proposed for the pRAM [1]. It has been shown [6] that a variant of this learning algorithm is particularly good at handling noisy data. The rein-

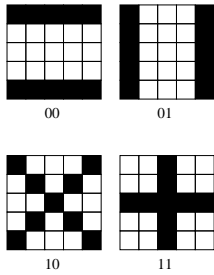


Figure 3:  $5 \times 5$  Patterns

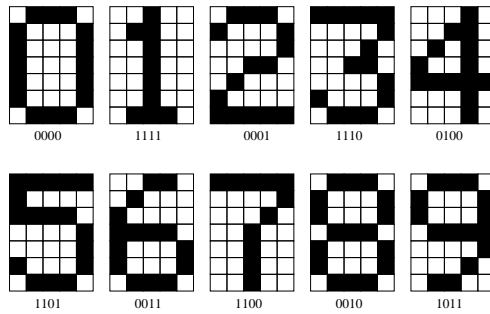


Figure 4:  $5 \times 7$  Patterns. As not all the 16 possible bit patterns out of four bits were used, the desired output was designed so that it is equally probable to have a ‘1’ or a ‘0’ in any position.

forcement learning rule gives the change in the firing probability  $\alpha_{\underline{u}}$  for memory locations  $\underline{u}$  is

$$\Delta\alpha_{\underline{u}} = \rho((a - \alpha_{\underline{u}})r + \lambda(\bar{a} - \alpha_{\underline{u}})p) \times \delta_{\underline{u},i}$$

where  $a$  is the state of the pRAM output,  $\bar{a} = 1 - a$ ,  $\rho$  and  $\lambda$  are parameters that determine the training rate,  $r$  and  $p$  are the reward and penalty signal respectively ( $r, p \in \{0, 1\}$ ). In a somewhat similar way to error back propagation [10], the contents of the memory locations that contributed towards the production of the output receive some adjustment. This tends to force these memory locations to adopt a value which is favourable towards producing the target outputs, and is guaranteed by the Kronecker delta  $\delta_{\underline{u},i}$  [1].

### 3 The Pattern Recognition Tasks

Pattern recognition tasks have been used in [1, 2, 6] to test the performance of the pRAM, with both noise-free and noisy patterns. For the purpose of this study, no noise is used during training.

#### 3.1 Symbol Recognition Task

For the symbol task [2], four images on a 5 by 5 matrix (figure 3) were used to train a pyramidal pRAM network. The network has five pRAM units in the input layer and two in the output layer giving a 2-bit binary output from the network. All the pRAM units have five input lines. Since five bits can address  $2^5 = 32$  different locations, there are a total of  $5 \times 2^5 + 2 \times 2^5 = 224$  memory locations in the network. Since each memory location contains an adjustable probability value, there are 224 adjustable parameters in this network.

#### 3.2 Digit Recognition Task

Similarly, for the digit recognition task [1], ten digits are represented on 7 by 5 matrices as shown in figure 4. The patterns are presented to a pyramidal pRAM network with five 7-input pRAM units in the input layer and four 5-input pRAM units in the output layer giving a 4-bit binary output. As not all the 16 possible bit patterns out of four bits were used, the desired output was designed so that it is equally probable to have a ‘1’ or a ‘0’ in any position. The pRAM units in the input layer are connected to all those in the output layer. The number of memory locations in the network is  $5 \times 2^7 + 4 \times 2^5 = 768$ .

## 4 Simulated Annealing

Simulated Annealing is a search technique that performs stochastic hill-climbing around a single point in the space of probable solutions (see figure 5). Initially, a random point is chosen, and it becomes the reference point. Next a new point is chosen randomly within a neighbourhood

1. Generate a random string  $x$
2. Initialise the starting temperature to  $T(0)$
3. Evaluate the string  $x$  by a function  $f(x)$
4. Make a copy of  $x$  giving  $x'$
5. Create a new string  $y$  by randomly adjusting a segment of  $x'$
6. Evaluate  $y$  by a function  $f(x)$
7. if  $f(y) > f(x)$   
     Discard  $x$  and make  $x$  becomes  $y$   
   else  
      $x$  becomes  $y$  with probability  $e^{((f(y)-f(x))/T)}$
8. if number of iterations exceeds the predefined counter change temperature  $T := r \times T$ , where  $r < 1$ ; and reset iteration counter to zero
9. Repeat steps 4-8 until  $T < T(*)$

Figure 5: Outline of the Simulated Annealing (SA) algorithm

of the reference point. This new point will become the new reference point either when it gives better result than the previous one, or stochastically otherwise. The probability of accepting a poorer point is controlled by the difference in performance between the two solution on hand and a *temperature* parameter which is progressively decreased. The gradually decreasing nature of the temperature makes it increasingly difficult for a poor performing point to be accepted. In theory, if allowed to run sufficiently slowly, SA is guaranteed to find the global optimal solution. However, for practical reasons, the search process cannot be run infinitely. Therefore, only a near-optimal solution is expected.

At each step, a neighbouring point is chosen by choosing a number (parameter) from the string, and adding or subtracting a small value to it. Protection against generating an invalid number outside the upper or lower bound is included in the algorithm which is the same as the mutation operation in the GA run to be described below. Details of the mechanism can be found in [8].

## 5 Genetic Algorithms

Genetic Algorithms (GAs) are machine learning algorithms based on ideas of survival-of-the-fittest and reproduction, borrowed from nature [7].

Typical combination of GAs and neural networks include the use of a GA as a connection weight training rule (in place of, say, error back propagation), a GA to determine the connectivity of a network which is subject to training by another learning rule, or both [12]. Work in these areas has studied both feedforward and recurrent multi-layer perception networks. This paper details our work using GA with pRAM networks.

Any problem to be tackled using GA must be translated into a string, or *chromosome*. Each slot in the string, or *gene*, represents a parameter in the problem to be optimised.

Evolving the probability value for all the pRAM units in a pRAM network is a challenging task for GA. This size of the solution space increases exponentially with the size of the string (i.e. as the number of parameters increases). The time needed to search the solution space thus increases with the chromosome size. For example, with a binary representation with strings of length  $l$ , the solution space equals to  $2^l$ . While binary digits are commonly used in GA strings, in this work real numbers are used since each memory location contains a probability value the range  $\{0,1\}$ . If we were to use binary digits we would have to convert between the binary representation and

1. Generate randomly a population of  $n$  strings
2. Evaluate all the strings in the population using  $f(x)$
3. Copy the best  $m$  strings to the new population
4. *Select* the best two individuals
5. From strings selected at 4, produce two new strings using *crossover*
6. Perform *mutation* on the new strings
7. Evaluate the new strings
8. Copy the new strings to the new population
9. Repeat steps 4-8 until the remaining  $(n - m)$  positions are filled up
10. Repeat steps 3-9 until all the strings gives the same result or when the maximum number of iterations allowed is reached

Figure 6: Outline of the Genetic Algorithm (GA)

the floating point number equivalent. Moreover, it has been reported [8] that a floating point representation produces results more quickly when it is used with appropriate specialised genetic operators.

The GA used here is similar to those described elsewhere [4, 5]. The genetic operations include *Selection, Crossover, and Mutation* (See figure 6 for the steps in detail). Selection is based on a *fitness* value associated with each string. The probability of a string being chosen is equal to its fitness relative to the total fitness of the population, and any string can be chosen more than once. This selection method is called *roulette wheel selection* that assign probability of being selected according to the individual's fitness relative to the sum of the fitness of all the individuals in the population. It is clear that there is a bias towards the better strings, i.e. those with higher fitness.

New strings, or *offspring*, are usually produced by copying part of the content of one *parent* - the selected strings, and fill the remaining gap from the other parent. The second offspring is formed similarly but picking up the remaining sub-strings. This process is called *crossover* and there are a number of different crossover schemes : the main difference between these is the number of sub-strings into which a parent is divided. In the experiments performed here, uniform crossover [11] is used, where crossover is independently decided on a gene-by-gene basis. This is to ensure maximum randomness in producing the new strings and thus maximum exploration into the solution space. In other words, it allows maximum disruption in the pairing up of 'good' genes. As the string size in the experiments are quite long, randomness is preferable in order to introduce diversity to the pool of strings.

Finally, the new strings are subjected to minor changes by a mutation operation so as to search the neighbourhood a feasible solution in close range. The one used here is *dynamic mutation* [8] which operates as follows. For a single gene  $x_i$  chosen at random from the chromosome  $x = (x_1, x_2, \dots, x_i, \dots, x_n)$ , a random binary digit is generated. A new gene,  $x'_i$ , is formed according to

$$x'_i = \begin{cases} x_i + \nabla(t, UB - x_i) & \text{if the random digit is 1} \\ x_i + \nabla(t, x_i - LB) & \text{otherwise} \end{cases}$$

where  $LB$  and  $UB$  are lower and upper bound on  $x_i$ , i.e.  $LB \leq x_i \leq UB$  ( $LB = 0$  and  $UB = 1$  in this study). The function  $\nabla(t, v)$  is defined by

$$0 \leq \nabla(t, v) = v * (1 - R^{(1-t/t_{max})^b}) \leq v$$

	5 x 5				5 x 7				
	Random	SA	GA <sub>400</sub>	Reinf	Random	SA	GA <sub>400</sub>	GA <sub>1200</sub>	Reinf
mean	25.0	62.75	78.49	99.55	6.25	21.72	26.09	83.78	99.79
s.d.	-	15.59	10.26	0.12	-	3.07	5.27	1.38	0.8

Table 1: The Recognition Rate (%) for the Digit Recognition Tasks

where  $R$  is a uniform random number in the range  $\{0,1\}$ ;  $b$  determines the degree of dependency on the iteration number;  $t$  is the current cycle number; and  $t_{max}$  is the maximum number of cycles to be performed.

In order to preserve ‘fitter’ strings in a generation, not all of them are replaced by new strings in the next generation. The best performing strings are copied directly without change to the next generation, leaving the rest places to be filled by the new strings. This scheme is called a *generation gap* [4].

## 6 Experimental Setup

### 6.1 Simulated Annealing

For the purpose of this simulation, the string contains floating point numbers within the range of  $\{0,1\}$ . This is to keep in line with the representation scheme used in the GA tested here. Binary digits were also tried but results suggest that neither shows significant advantage over the other.

Experimental results suggest that SA is very sensitive to the initial temperature value  $T(0)$ . If the initial temperature is too high, it may cause the SA algorithm to accept almost all the points as new reference point. In contrast, if the temperature is too low, it may make it difficult to swap over to another point. In the experiments performed, the starting and ending temperatures are set to 0.01 and 0.00001 respectively with a reduction factor of 0.9 every 500 cycles. The performance measurement is the recognition rate which is in the range  $\{0,1\}$  (shown as percentage of recognition in Table 1).

### 6.2 Genetic Algorithms

In most of the experiments performed, the population size is 400. As the chromosomes are very long (244 or 768 real-valued parameters), a large population is used here to try to provide sufficient diversity. The probability of performing crossover and mutation are set to 0.75 and 0.1 respectively. A generation gap of 0.3 is maintained [4]; the best performing strings (30%) of every generation are passed to the new generation, while the remaining space (70%) is filled with new strings. The fitness (performance) measurement used is the inverse of the mean squared error.

For the ‘5 x 7’ task, an additional arrangement was also used in order to try to achieve better results. The population size in the second setup was increased to 1200 with crossover and mutation rates of 0.6 and 0.5 respectively. The population size and the mutation rate are set exceptionally high with the objective of introducing diversity into the population.

Experimental results are given in Table 1 under the columns GA<sub>400</sub> and GA<sub>1200</sub>.

## 7 Results and Discussion

In a completely random situation, the probability of producing the expected output of  $q$  bits is  $p = (1/2)^q$ . Thus, the mean recognition rate for are 25% and 6.25% for the symbol task (two-bit output) and digit task (four-bit output) respectively. These are the lower bounds for comparing the performance of the learning algorithms to be tested. Any algorithm that learns should certainly perform better than this.

The recognition rates reported in table 1 are averaged from 5 runs. In both cases, the GA<sub>400</sub> performed better the SA with 95% confidence. Both performed significantly better than the

minimum benchmark value (column ‘Random’) showing that some degree of learning has been achieved. However, neither of these are as good as the original reinforcement learning algorithm.

It is not surprising to find that both algorithms tested here do not perform as well as the reinforcement algorithm. The updating procedure of the reinforcement algorithm is tailored to change only those memory locations responsible for producing the output. However, both the SA and GA algorithms do not have any *a priori* information about the search space. Moreover, some memory locations are not used by any patterns, and it is highly probable that the SA and GA algorithms are modifying the unused location a lot of the time. This will not only prolong the search time but also mislead the algorithms when they are deciding which solution to keep or discard. Reinforcement learning is undoubtedly a more effective and efficient method.

In terms of robustness, GA not only performed better, but also is more stable (smaller standard deviation) than SA in the ‘ $5 \times 5$ ’ task. The drastic drop in performance for both SA and GA in the ‘ $5 \times 7$ ’ task might be due to the increase in the string size (768 vs 224). The GA<sub>1200</sub> run (with population of 1200) give a result as good as 83.78% recognition rate. This indicates that there may be insufficient diversity in the GA<sub>400</sub> setup. The drawback here is that the search time increased more than three times, and further simulations will be required to explore possible improvements in this direction.

It should be noted that the SA and GA learning algorithms are not straightforward to use since there are a number of parameters which need to be fine-tuned for best performance. This is a significant disadvantage in the use of these algorithms. Moreover, the time needed for both algorithms to find the optimal solution increases drastically with the size of the search space, making them less useful for solving bigger problems.

In order to make SA and GA to perform as well as the reinforcement learning algorithms, *a priori* information will have to be injected into the system. One way to achieve this is by counting the number of memory locations actually accessed by the training patterns, which is likely to be less than the total number of memory locations. This will effectively reduce the size of the string(s) used in the SA and GA search. As a result, searching will be bounded within a “subspace” where the optimal solution resides.

Another feasible direction for the GA is to adopt a radically different representation scheme. Instead of using the firing probabilities in the chromosome of a GA, these could contain paths from input layer to output layer. Only the memory locations on these paths would contain non-zero firing probabilities. This would avoid adjusting unused locations. Domain-specific information can also be learned by the GA during the run when poor performing paths are discarded. More work is needed in this direction.

## 8 Conclusion

We have compared Simulated Annealing (SA), Genetic Algorithms (GAs) and reinforcement learning to train pRAM networks. GAs perform consistently better than SA, particularly when a very large population is permitted.

However, neither SA or GA perform as well as the original reinforcement learning algorithm developed for pRAM networks. The main reason for this is likely to lie in the availability of domain specific information in the case of the reinforcement algorithm.

This study also indicates that both SA and GA suffer seriously when the size of the problem increases. It is not clear at this moment whether larger population size guarantees better performance. This has to be confirmed by further experiments.

## 9 Acknowledgments

K.C. Tsui is partly supported by the Rotary Foundation of Rotary International through the provision of 1994-1995 Rotary Ambassadorial Scholarship. Dr. Plumley is partly supported by grant (GR/J383987) from the UK Science and Engineering Research Council. Test program for the reinforcement learning algorithm is kindly provided by Dr. Y. Guan.

## References

- [1] T. G. Clarkson, D. Gorse, and J. G. Taylor. Biologically plausible learning in hardware realisable nets. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, pages 195–199, 1991.
- [2] T. G. Clarkson, C. K. Ng, and Y. Guan. The pRAM: An adaptive VLSI chip. *IEEE Transactions on Neural Networks*, 4(3):408–411, 1993.
- [3] T. G. Clarkson, J. G. Taylor, and D. Gorse. pRAM Automata. In *Proc. IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA '90)*, pages 235–243, 1990.
- [4] K. A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD Dissertation, University of Michigan, 1975.
- [5] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, 1st edition, 1989.
- [6] Y. Guan, T. G. Clarkson, J. G. Taylor, and D. Gorse. A noisy training method for digit recognition using pRAM neural networks. In *Proc. IEEE International Joint Conference on Neural Networks*, pages 673–678, 1992.
- [7] J. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 2nd edition, 1992.
- [8] C. Z. Janikow and Z. Michalewicz. An experimental comparison of binary and floating point representations in genetic algorithms. In *Proceedings of Fourth International Conference on Genetic Algorithms*, pages 31–36, 1991.
- [9] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.
- [10] D. E. Rumelhart, , G. E. Hinton, and R. J. Williams. Learning internal representation by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, volume 1, chapter 8, pages 318–362. MIT Press, 1986.
- [11] G. Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of Third International Conference on Genetic Algorithms*, pages 2–9, 1989.
- [12] L. D. Whitley and J. D. Schaffer, editors. *Proceedings of International Workshop on Combinations of Genetic Algorithms and Neural Networks*. IEEE Computer Society Press, 1992.