A review of RAM based neural networks

J Austin
Department of Computer Science
University of York
York, YO1 5DD, UK.

Abstract

This paper briefly reviews a class of neural networks often termed as RAM based networks. As this paper shows, the networks are identified by their use of 'logical' 1-in-n decoders as a pre-process to each neuron. The paper explains why the networks have also been termed weightless systems. Two sub-classes of binary neural networks are described, those which use binary weights and use only a single layer of neurons consisting of the MRD, ADAM and WISARD, and those which use multi-valued weights and multiple layers of neurons comprising the PLN, pRAM, GSN, TIN. The paper attempts to show the evolution of the networks, as well as describing the benefits of this class of neural network for hardware implementation.

1 The motivation for binary neural networks

The typical model of a neuron used in a large number of neural networks is based on the McCulloch and Pits [Pit43] neuron which can be described by equations (1) and (2). These specify a linear weighted sum of the inputs, followed by a non-linear activation function.

$$u = \sum_{j=1}^{n} w_j x_j \tag{1}$$

$$y = \frac{1}{1 + e^{-au}} \tag{2}$$

Where u is the activation of the neuron, w are the weights, x are the inputs, a controls the shape of the output sigmoid, and y is the output. When equation (2) is replaced by a Heavyside function, the neuron is called a Linear Threshold Unit (LTU).

When given the appropriate activation function and used in networks with 3 layers (MLN) they have been shown to be universal function approximators [Zur92]. In addition, they have very good generalization abilities. However, this universality comes at a cost. To train a MLN an arbitrary complex problem requires repeated presentation of training examples, which often result in very long learning times.

The implementation of the feed forward operation of the networks requires the use of a multiplier unit, which can be problematic for fully parallel VLSI implementations of the networks, due to the large size of the multiplier. The most popular approach to training is the generalized delta rule, and its derivatives [McC86] which requires both the forward propagation phase and a backward (back error) propagation phase. The complexity of the training algorithm has limited its implementation in fully parallel dedicated hardware.

2 N tuple method

The RAM based systems were not originally designed with a consideration of the limitations of MLNs, but do provide solutions to these problems. They originate in the work of Bledsoe and Browning [Bro59], who invented a method of pattern recognition commonly termed the N-tuple method. The basic principle behind the N tuple method is that learning to recognize an image can be thought of as building a set of logic functions that can describe the problem. The logic functions will evaluate true for all images which belong to the class that the logic function represents and evaluate false for all other classes.

The is shown in the simple example in Fig. 1. Each class of image has a set of logic functions that relate to it. For an unknown image, the set of logic functions that has the majority of functions which evaluate to true, indicate the class of image. The image of a T can be recognized by using the logic function;

$$R = A.B.C + \bar{D}.E.\bar{F} + G.H.I \tag{3}$$

0-8186-6710-9/94 $4.00

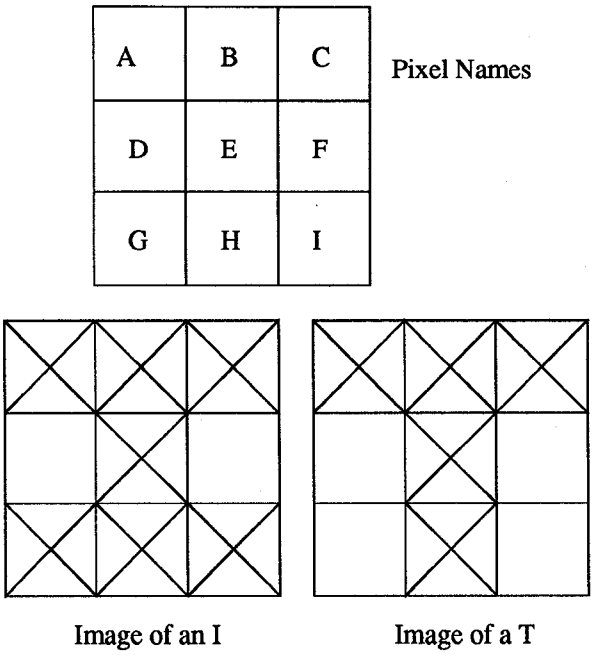

Figure 1: This is an example of the N tuple process.

The image of an I can be recognized by using the logic function;

$$R = A.B.C + \bar{D}.E.\bar{F} + \bar{G}.H.\bar{I} \qquad (4)$$

To improve the generalisation ability of the N tuple method, instead of logically ANDing the minterms of the expressions together, they are arithmeticaly summed to give a count of the number of terms that are true.

In the example in Fig 1 there are 3 'tuples' each of size 3. These form the minterms in the equations above. They are

Tuple 1; pixels A B C

Tuple 2; pixels D E F

Tuple 3; pixels G H I

To learn the logic functions that represent the data belonging to a given class was originally shown by Aleksander and Stonham [Sto79]. The approach was based upon the structure shown in Fig 2. The problem involves remembering which logic terms would be needed for specific classes of image. This was most easily achieved by using a logical 1-in-N decoder followed by a set of binary storage locations for each term, and using one such unit for each tuple. The logical decoders compute all possible logical functions of the N inputs they connect to. When presented with a pice of data, the various decoders will indicate the functions required. To recognize an unknown piece of data, the same approach is used, only now the storage locations are accessed and summed.

The definition of a RAM node is one tuple of N inputs, followed by a logical 1 in n decoder, followed by a set of storage locations and a summing device. This is shown in fig. 2.

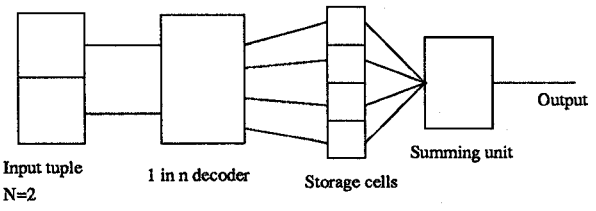

Figure 2: The basic RAM node.

The 1 in n decoder has the truth table shown in table 1.

Input A	Input B	Output lines
0	0	1000
1	0	0100
0	0	0010
0	0	0001

Table 1: Activation table for 1 in n decoder

2.1 RAM based units compared to LTU

In comparison with conventional linear threshold units (LTU), RAM units contain a greater amount of classification power. They can be described using LTU, but would require a 3 layer network to achieve this.

RAM based units are most similar to 'higher order' networks [Pao89], which combine the input data prior to the inputs application to a system composed of LTUs. However, where higher order units combine continuous values using non-linear functions (powers etc.) RAM units combine the inputs using logical functions (AND and OR). In effect, this is the same approach, only one is continuous the other is binary.

Fig. 3 and 4 show the relation between a network of RAM units and a higher order network. Fig. 3 shows what is commonly called an N tuple network. One such network can be used to identify the similarity of an unknown image to one trained. A group of such networks, used to recognize one of a number of classes is called a Multi-RAM discriminator (MRD). Fig. 4 shows a higher order network with an equivelent structure as an N tuple network.

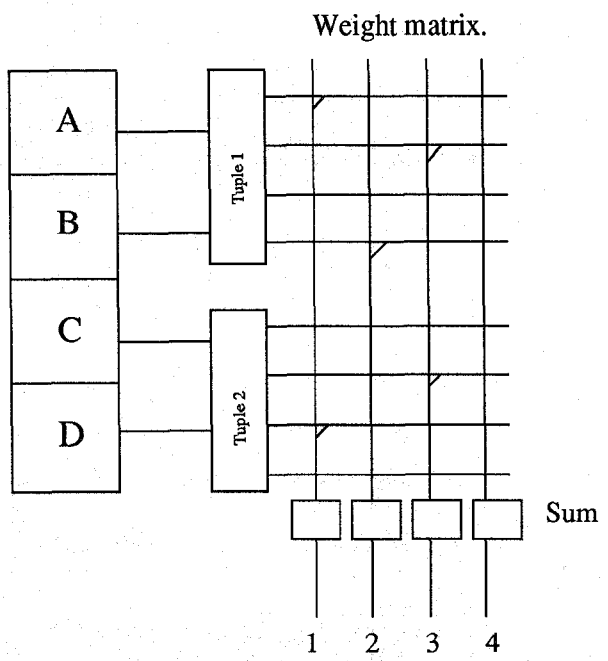

Figure 3: The N tuple network.

Another important difference is that RAM units always take sub-samples from the image in the form of tuples. Each tuple is made up of N samples of input data, fed to its own set of storage cells. In RAM based systems, the size of a tuple, N, is an important parameter as it effects the classification ability of the network.

The major advantage of the RAM based approach is that the decoder/storage cell combination is a random access memory (RAM). Thus allowing very simple and direct implementation in cheap and readily available components. This was shown by Aleksander et. al. [Bow84] in the WISARD pattern recognition machine.

2.2 The limitations of the N tuple method

Although the basic N tuple approach was powerful, in terms of is learning speed and simple implementation. The method has a major limitation. This relates to the learning capacity of a given N tuple network. By inspection it may be obvious that a given N tuple network cannot implement all possible functions of the data inputs.

To show this a simple problem (intra-exor) problem is shown in table 2 (from [Aus93]).

The tuple distribution given in fig. 5 shows a two tuple system that cannot solve the problem intra-exor problem shown in table 2 with a given N tuple network. However, by altering the placement of each tuple as shown in fig 6, the problem is solvable. This is unlike the EX-OR problem when used on a single layer network. The network cannot solve the problem given an arbitrary ordering of the inputs.

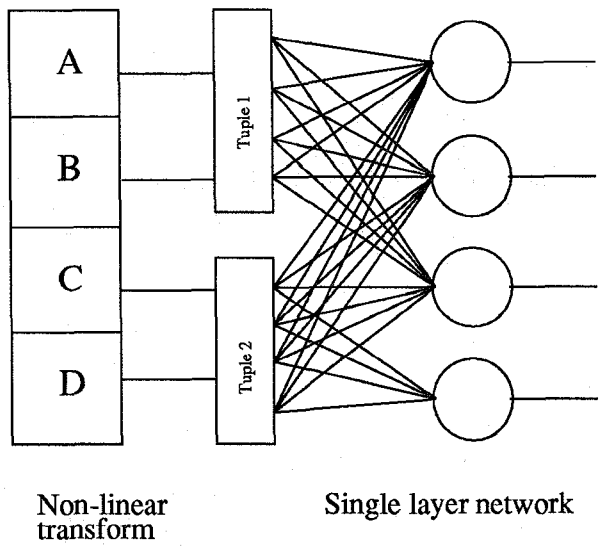

Figure 4: A single layer net with N tuple preprocessing.

Inputs ABCD	Output
1 0 1 0	1
0 1 0 1	1
1 0 0 1	0
0 1 1 0	0

Table 2: The intra-exor problem

To solve this problem, one could used an tuple size equal to the input data size. However, this looses any generalization ability of the network, and results in RAM nodes with a very large memory requirement.

The N tuple method was used in the WISARD pattern recognition machine [Bow84], capable of recognizing images at about 25 frames per-second. To overcome the problem given here, the training and testing method involved moving the image around whilst training and testing. This effectively ensured that most of the time the EXOR type problem did not arise (it also allowed good generalization).

For many problems, such as image processing, the intra-exor problem is not an issue, as training methods overcome the limitation. The method has been successfully applied to many problems, such as monitoring crowded underground railway platforms, face recognition [Ale85a], satellite image recognition [Buc94] and character recognition [Row93].

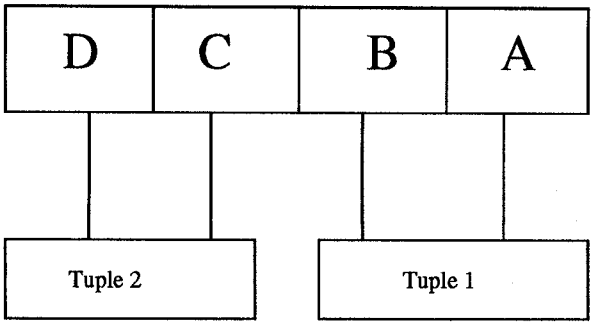

Figure 5: An arrangement of tuples that will not solve the intra-exor problem

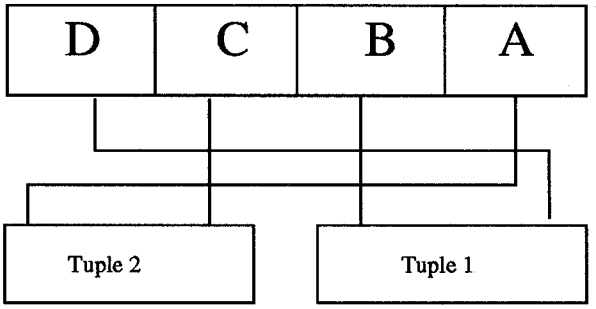

Figure 6: An arrangement of tuples that will solve the intra-exor problem

A number of extensions to the basic N tuple method have been developed. To deal with non-binary inputs Wilson [Ale85b] and Austin [Aus88] have developed methods for taking in grey scale data and still allow the use of binary logic functions.

2.3 Hardware implementation of the N tuple method

The N tuple method was implemented in dedicated hardware in the form of WISARD in the early 1980's [Bow84]. The machine was taken up commercially by Computer Recognition Systems of Woking (UK) and marketed for a number of years. The commercial system used conventional digital processor methods and achieved a recognition rate of 12.5 frames a second for images of 512^2 images, using sparse sampling (i.e. not all pixels used).

More recently the method has been used in a parallel image processing system (C-NNAP,[Pac94]), where the method has been extended to form an associative memory [Sto87]. This system uses Field Programmable gate arrays to implement the memory scanning and training processes.

3 Overcoming the limitations of N tuple; RAM pyramids

As the intra-exor problem has shown, the use of the N tuple method can be rather hit-or-miss. Some training examples will not be separable, while others will be.

To increase the robustness of the method the functional capacity needed to be raised whilst maintaining the generalization ability, along with the training speed, and simple hardware implementation.

To achieve this, it is important to understand how the method is capable of generalizing on unseen data. The method operates by a set membership classification process. Each image is broken up into a number of tuples. In the N tuple method, for a given set of patterns the binary patterns appearing in each tuple are recorded during training. During testing, each tuple is checked by each RAM node to see if it contains a known bit pattern, and the number of RAM units that recognize the input tuple pattern is counted and output as a recognition figure. In effect each RAM unit, which process the tuple data is looking for patterns that belong to a set of known patterns that were presented during training. The generalization comes from the way tuple patterns are allowed to be mixed between training examples. The larger the tuple size the smaller will be the generalization set size.

Unfortunately, the size of the generalization set is not set by training, (apart from the number of examples given), but by the tuple size, N. Thus, to get a good balance between classification success and generalization, requires experimentation.

The problems are caused by the linear combination of the results of the RAM units. In the N tuple method these are summed. This results in the intra-exclusive OR problem given in section 2.2.

The obvious solution is to combine the outputs of the RAM units non-linearly. This can be done in two ways; use a multi-layer network of linear threshold units or use more RAM units. The former method has been called a Hybrid network [Aus93]. This is very similar to the higher order networks, except that it uses logical functions to combine data. The approach results in a solution to the problem, but at the cost of longer training time (iterative). The latter methods are more popular, and uses the RAM based approach exclusively.

The typical form of the multi-layer RAM net (MLRN) is to combine the results of one layer of RAM units using subsequent layers. Because each RAM unit has a limited number of inputs, it is necessary to have

many layers of RAMs for large images.

The typical form of a MLRN is shown in Fig. 7.

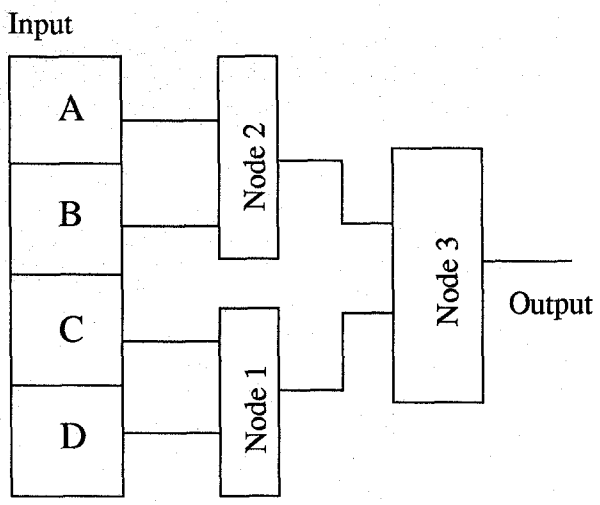

Figure 7: Typical form of a multi-layer RAM network.

These networks can implement any subset of logical combinations of the input data [Sto92b]. Thus, overcoming the intra-exclusive OR problem given earlier. However, the solution requires a new training method, for the same reason that the multi-layer networks of LTU required the introduction of the generalized delta rule (GDR). As was shown by the GDR, to perform back propagation requires soft-limited output functions on each neuron. The function implemented by RAM units is not continuously differentiable, thus the MLRNs cannot be trained using the GDR when implemented using the basic RAM node. This problem has forced a number of researchers to investigate how MLRN can be trained. Notably, the Probabilistic Logic Node (PLN) [Ale89], the probabilistic RAM (pRAM) [Tay93], the Goal Seeking Neuron (GSN) [Bis91] and Time Integrating Neuron (TIN) networks [Gur92], and their derivatives.

All these networks provide solutions to training MLRNs.

3.1 Reinforcment Learning in Multi-layer RAM nets - the PLN

Because multi-layer RAM networks cannot be trained using a back-propagation like algorithm, reinforcement learning method has been used, which does not require a direct measure of error, but just an indication that the output was incorrect. The reinforcement signal is sent to all nodes and used to determine if the pattern present on the input to the RAM unit should be saved (i.e. the logic function noted).

As the RAM node stands, reinforcement could not be used. This is because the node can only record (1) the presence of a particular binary pattern during training, or (2) the absence of the pattern. This binary representation would not work with reinforcement learning as three states are needed for the method to work, namely (1) the pattern input caused a correct output, (2) the pattern input caused an incorrect output, (3) the pattern did not occur on the input. By allowing tri-state storage in the RAM node reinforcement learning could be used. This was implemented by Aleksander in the PLN [May92]. In practice the PLN represents the information as;

1 = tuple pattern occurred and is correct,

0 = tuple pattern occurred and is incorrect,

u = pattern has not occurred.

The use of the u, 'don't know' state required an extension of the learning algorithm. When a 'u' state is accessed in the RAM, the output of the ram is set to 1 or 0 with a probability of 0.5. Thus any input pattern will allow the propagation of a result to the output of the network.

The algorithm used to train this version of the PLN, is as follows;

1) Initialize all locations to a random binary [1,0] values.

2) Select an input pattern.

3) Access the RAM and generate an output.

4) If the value at the output of the net is correct, set the reinforcement signal, r, to +1. If the output is incorrect, set the reinforcement signal to -1.

5) for all nodes, If r=+1 and the node is addressing a 'u', then set the node to the value set on the output of the node.

6) If r=+1 then set the next input pattern to be learned.

7) If r=-1, then clear all the nodes and re-enter input pattern.

8) goto 3.

The algorithm requires repeated application of this process, until all nodes have learned the patterns.

The approach taken in the PLN required (1) iterative learning, (2) 3 state storage locations. As a result, direct implementation in standard RAM components is not possible and training is slower. However, training is still more rapid than other approaches using LTUs.

3.2 Weighting the storage locations, N tuple RAM extensions and the w PLN

The approach used by the PLN nets and by the N tuple approach only permitted each node to record the presence and absence of a particular input pattern. This approach can be softened by allowing the system to record the frequency of occurrence of the input patterns to a RAM node. This would allow the important features of a piece of data to be weighted in preference to other features.

The original N tuple method described by Bledsoe and Browning suggested this. However, because of implementation difficulties it was not used in the hardware implementations of the N tuple method. It has been shown that the approach can improve recognition accuracy [Bro59]. To overcome the implementation problem Austin and Smith used a weighted scheme during training and converted this to a binary representation for later implementation in RAM based nodes [Aus92]. This approach retained the improved recognition accuracy while partially maintaining the implementation efficiency.

In the case of MLRN, Mayers [May92] with the w-PLN and Gorse and Taylor [Tay93] with the pRAM, both realised the benefit of using a weighted scheme in MLRNs. The pRAM is described in the next section. Mayers developed the w-PLN which generalized 3 state storage used in the PLN to w states, and showed the improved recognition accuracy which resulted. The reinforcement algorithm was extended to take this into account. This approach has not been implemented, but has been used to model delay learning in invertebrates [May92].

Nevill and Stonham [Sto92a] provide a description of the PLN which has the addition of a sigmoid activation function in a w state PLN. In this one, the storage locations hold values with w states. However, after the value is addressed and read out, the value is passed through a sigmoid normalization function which gives a continuous value between 0 and 1. This value is interpreted as the probability that the unit will fire with a 1. The same reinforcement algorithm is used.

3.3 Storing probabilities in the RAM locations, the pRAM

Gorse and Taylor [Cla94] fully extended the design of a RAM unit to a probabilistic framework. The probabilistic RAMs hold the probability of a given input pattern occurring, instead of just holding a normalized value as done in the PLN approach. By using probabilities, the likelihood of the input pattern belonging to a particular class can be calculated, rather than a boolean yes/no class membership decision.

In addition they pass the probability accessed by any input pattern to other nodes. Thus, the approach is purely probabilistic in its operation.

To implement this approach, the Kings' team have used a fully probabilistic approach, where information is passed between the nodes probabilistically. To do this they introduced a pulse coded method of communication between RAM nodes [Tay89] which simplifies the operation and hardware implementation of the node [Gua93]. For each cycle of operation of the pRAM, the node (1) determines if pulses are present at its inputs, (2) forms a binary pattern of the bits that are on and off, (3) accesses the memory location that is addressed by that bit pattern (4) reads out the probability, u, that the node fires from the memory location. (5) sets a one or a zero at the output of the node depending on u. Thus, over time the node will access a number of memory locations and, over time, fire with a mean rate depending on those nodes values. This is an entirely probabilistic system, including the hardware implementation, which makes it unique amongst neural network systems. The pRAM uses the reinforcement learning method to update the probabilities.

3.4 Non-reinforcement learning and probabilistic RAMs - the GSN

Another way of implementing a probabilistically based RAM node is not to use continues firing rates, but to pass the probability of the node firing using a boolean value. This method, used in the GSN [Bis91] allows the system to work statically, i.e. at any moment the exact probability of a nodes output can be obtained. In the pRAM, to obtain the probability of the output requires an averaging of the output pulses from the neuron. In addition, the GSN does not use reinforcement learning, but a method that allows learning of an unknown pattern in one presentation. The method uses a form of back propagation and basically operates as follows, using a pyramid of RAMs. In the forward pass phase (validation), the aim is to see if the network can potentially output the correct result. Each node (RAM) receives an input from the image which causes access to a memory location in each RAM. Initially all RAMs contain the undefined state, u. Thus all units n the first layer output this state. This value is passed to the next layer of units. These units must then interpret the u state. In the GSN, a u on an input address line is taken to be 1 or

0 input. If a node receives the input data 1,1,u,u this implies that the unit (with, for example, 4 inputs) must access locations 1,1,1,1 and 1,1,1,0 and 1,1,0,1 and 1,1,0,0. The node must then resolve these conflicting inputs. It holds these as a list and selects one to use to access the memory location. It then passes contents of the addressed location to the next layer. The value passed to the next layer (the final layer) may also be a 'u', which means that a list of possible inputs will be formed at the final node. The same process of selecting one of these is performed. If this unit outputs the correct value (1 or 0), then the chosen patterns at each node which has u's on its input is correct. These patterns are selected by training the node (changing the accessed u value to the chosen output value). If the final layer node outputs 'u', the same process operates, but now the final unit is set to the correct output. If the final unit outputs the incorrect value, then the nodes which have a selection of possible inputs are revisited and a different input pattern chosen. By a careful search process the network is trained.

Although the GSN operates in 'one pass', it embodies a search process that is similar to a depth-first search with backtracking in AI. This search process has a large worst case search time. So, although the training set is only presented once, each pattern can take some time to learn. Furthermore, the network may not train on one pass, as the ordering of the patterns may result in the system not finding a solution.

3.5 The crossing the divide, the sigma-pi unit

The sigma-pi unit[McC86] has been shown to be equivalent to a RAM based node by Gurney [Gur92]. He shows how a RAM based node can be made equivalent to the sigma-pi units and shows how, with the addition of systems that perform a sigmoid activation on the output of a node, so that the network can use back propagation learning. He shows that, if a node is described by the following,

$$a = \frac{1}{S_m 2^n} \sum_u S_u \prod_{i=1}^{n} (1 + u_i x_i) \tag{5}$$

$$y = \sigma(a) \tag{6}$$

Where S_m is the range covered by the values stored at each addressed location, n is the number of input values (tuple size), S_u is the value held in the addressed location, u is one of the RAM addresses, and x is the input tuple pattern. The term $\prod_{i=1}^{n}(1+u_i x_i)$ effectively activates a given address if the tuple matches the indexed address, and the term $\sum_u$ indexes all addresses in the RAM. The final equation is a sigmoid activation function σ, and the output of the neuron is given by y.

Then the operation of a RAM can be made continuous over its inputs and its outputs. As he points out, the practicality of this model is bound by the the size of n (the number of bits in the input). He then proposes a stochastic version of this model (TIN) which reduces the computational complexity. The approach is similar to that taken in the pRAM, using bit streams as inputs and outputs of the unit. However, his model incorporates a sigmoid output function which makes the node continuous. The result of this is a node that can be interpreted as a continuous system and, as Gurney shows, can be trained using a version of back error propagation. However, it is not clear if any advantage is gained from this in terms of hardware implementation or speed.

The hardware implementation of this node type has been described by Hui, Morgan, Gurney and Bolori [Bol92]. There most recent chip implements 10,240 neurons in ES2 1.5um double metal CMOS.

4 Concluding remarks

The development of the RAM based approach has come a long way since the original concept was proposed by Bledsoe and Browning in 1959. The original RAM implementation is still the simplest to implement in dedicated hardware, with theoretical processing speeds in the order of 10's of nano seconds for both training and testing. The power of the approach has been demonstrated in the ADAM system and the grey scale extensions to the method. The speed of the method was traded off against robustness, in that all training patterns may not be learned by the system. The MLRN's overcame these problems allowing complex problems to be learned without the possibility of the network failing on any examples. The original PLN showed how reinforcement learning could be used in RAM networks, requiring only the addition of a one extra state in the nodes memory locations. Increased performance has been gained by the use of multi-state w-RAMs, but at the cost of higher implementational complexity. The pRAM allowed simpler hardware implementation as well as adding better classification ability and a clear biological interpretation. Both the pRAM and the PLN required the addition of iterative training, which breaks

with the one shot training ability of the original N tuple method. To overcome this the GSN introduced a scheme where patterns where only presented once for training, but introduced a search process in each training cycle. This allowed on-line training and up-date to a network. Finally the TIN neuron and its derivatives clearly showed that the RAM unit could be trained using conventional methods if certain variations were added.

Four major implementations of the RAM based systems exist. The original WISARD (discrete components), the ADAM multi-processor (C-NNAP, FPGA), the pRAM (VLSI) and the Hui et al. (VLSI) implementation. All have shown that hardware implementation of the learning algorithms is feasible, and that many potential applications can benefit from the hardware implementation of these methods.

The RAM based neuron undoubtedly provides a novel approach to the design of neural network systems, which provide a range of techniques for many applications. The most useful addition to future research in RAM based systems would be a comparison of the methods against the performance of other networks on a set of standard benchmark problems.

5 Summary

This paper has briefly described the work in the area of RAM based neural networks. It has shown that these neurons all have a non-liner function prior to the weights, which provides the rapid training characteristics. In addition, when the binary version of the nodes are used, the implementation of the decoder and storage is simply a logical decoder and set of bit storage locations.

To some extent, the simple implementation characteristics have been lost in later improvements to the RAM nodes. However, this is the cost to be paid to obtain better and more reliable classification ability. The use of a reinforcement learning scheme has provided methods that can be implemented more simply than systems that require heavy iteration between the neurons.

References

[Ale85a] I. Aleksander. Wisard: A component for image understanding. *IEEE Proc.*, 135 Pt.E(5), 1985.

[Ale85b] M J D Wilson I Aleksander. Adaptive windows for image-processing. *IEE Proceedings-E Computers and Digital Techniques*, 132(5):233–245, 1985.

[Ale87] W K Kan I Aleksander. A probabilistic logical neural network for associative learning. In *Proc. of IEEE 1'st anual Conference on Neural Networks*, pages 541–548, San Dieago, 1987.

[Ale88] I Aleksander. Logical connectionist systems. In R Eckmiller C von der Malsberg, editor, *Neural Computers*, pages 189–197. Springer Verlag, 1988.

[Ale89] I Aleksander. Cannonical neural networks based on logic nodes. *IEEE International Conference on Artificial Neural Networks*, pages 110–114, 1989.

[Aus88] J Austin. Grey scale n tuple processing. In Josef Kittler, editor, *Lecture Notes in Computer Science:301*, pages 110–120. Springer-Verlag, 1988.

[Aus92] G Smith J Austin. Analysing aerial photographs with adam. In *International Joint Conference on Neural Networks*, Baltimore, USA, June 1992.

[Aus93] J Austin. Rapid learning with a hybrid neural network. *Neural Network World*, 5:531–549, 1993.

[Bis91] E C D B C Filho M C Fairhist D L Bisset. Adaptive pattern recognition using the goal seaking neuron. *Pattern Recognition Letters*, 12:131–138, 1991.

[Bol92] T Hui P Morgan K Gurney H Bolori. A cascadable 2048-neuron vlsi neural network with on board learning. In *ICCAN'92*, pages 647–651, 1992.

[Bow84] I. Aleksander W. V. Thomas P. A. Bowden. Wisard:a radical step forward in image recognition. *Sensor Review*, pages 120–124, July 1984.

[Bro59] W W Bledsoe I. Browning. Pattern recognition and reading by machine. In *Proc. Joint Comp. Conference*, pages 255–232, 1959.

[Buc94] J Austin S Buckle. The practical application of binary neural networks. In *Proc. of the UNICOM seminar on Adaptive computing and information processing*, 1994.

[Cla94] D Gorse J G Talor T G Clarckson. Extended functionality for prams. In *ICCAN 94*, pages 705–708, 1994.

[Gua93] T G Clarkson C K Ng Y Guan. The pram: An adaptive vlsi chip. *IEEE Transactions on Neural Networks*, 4(3):408–412, May 1993.

[Gur92] K N Gurney. Training networks of hardware realisable sigma-pi uints. *Neural Networks*, 5:289–303, 1992.

[May92] C E Mayers. *Delay Learning in Artificial Neural Networks.* Chapman and Hall, 1992.

[McC86] D E Rummelhart J L McClelland. *Parallel Distributed Processing.* MIT Press, 1986.

[Pac94] J Austin A Turner S Buckle M Brown A Moulds Rick Pack. The cellular neural network associative processor, c-nnap. *IEEE Computer*, To be Published. 1994.

[Pao89] Yoh-Han Pao. *Adaptive Pattern Recognition and Neural Networks.* Addison-Wesley, 1989.

[Pit43] W S McCulloc W Pitts. A logical calculus of the ideas imminent in nevous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[Row93] R Tarling R Rowher. Efficient use of training data in the n-tuple recognition method. *Electronics Letters*, 29(24):10932094, September 1993.

[Sto79] I. Aleksander T. J. Stonham. Guide to pattern recognition using random-access memories. *Computers And Digital Techniques*, 2(1):29–40, 1979.

[Sto87] J Austin T J Stonham. An associative memory for use in image recognition and occlusion analysis. *Image and Vision Computing*, 5(4):251–261, Nov 1987.

[Sto92a] Neville T J Stonham. Adaptable reward penilty for pln. In *ICCAN'94*, page 631, 1992.

[Sto92b] R Al-Alawi T J Stonham. A training stratagy and functionality analysis of digital multi-layer neural networks. *Journal of Intelligent Systems*, 2(1-4):53, 1992.

[Tay89] D Gorse J G Taylor. An analysis of noisy ram and neural nets. *Physica D*, 34:90–114, 1989.

[Tay93] D Gorse J G Taylor. Review of the theory of prams. In N M Allinson, editor, *Weightless neural network conference*, pages 13–17. University of York, 1993.

[Tay94] S Ramanan R S Peterson T G Clarkson J G Taylor. Adaptive learning rate for training prymidal prams. In *ICCAN'94*, pages 1360–1363, 1994.

[Zur92] Jacek M. Zurada. *Introduction to Artificial Neural Systems.* West publishing co., 1992.