P pattern recognition based on a probablistic RAM net using *n*-tuple input mapping

M.Ouslim
K.M.Curtis

Indexing terms: Digital neural network, pRAM, n-tuple, Pattern classification, Node connectivity

Abstract: A multilayer digital neural network, based on the probabilistic random access memory (*p*RAM), is used as a *P* pattern classifier system. This network presents an elaborate implementation of the *n*-tuple technique, which has mostly been used for pattern recognition (Bledsoe and Browning, 1959). The network's main properties, discrimination and generalisation, are discussed as a function of the *p*RAM connectivity. Pyramid networks, based on different *p*RAM connectivities, are simulated using an enhanced version of global reinforcement learning. *n*-tuple input mapping based on data analysis is proposed. The results show that combining the permuted data-based input mapping with a *p*RAM net, using different node connectivities through the pyramid layers, can achieve a good balance of the network's properties, when handling a *P* pattern classification task. Results are presented for the 10 digit recognition problem, which are motivating and very encouraging.

1 Introduction

The *n*-tuple technique addresses the problem of extracting general features in a pattern recognition task. In the case of images, it consists of handling the whole image as a set of its parts, each of which is a group of *n* pixels called an *n*-tuple [1–4]. Studies into the *n*-tuple technique have led to new types of Boolean neural networks. The alternative to implementing these neural networks, based on random access memories (RAMs), presents an attractive and cost-effective solution. Different variants of the RAM exist; they differ by the type of information they can store and the type of strategies governing their functionality within the network. Among them we cite the probabilistic logical node (PLN) [3] and the goal seeking neuron (GSN) [5], which is a RAM with a storage capacity of three values 0, 1 and **u** (undefined), and the probabilistic RAM (*p*RAM) [6], in which a continuous memory word is used. The *p*RAM generalises the Boolean neuron concept; this is why it is used as a basic node in this paper.

The Boolean neuron connectivity affects the properties of the net in a conflicting manner; a small connectivity enhances generalisation but degrades discrimination, whereas a high connectivity was shown to achieve the reverse [2].

In this paper, we describe the use of a *p*RAM net arranged as a pyramid (reverse tree) [7, 8], where the nodes present different connectivities in an attempt to balance between the net's main properties. In addition, in contrast to previous work [5, 7–10] which has been devoted to enhancing the *n*-tuple processing stage, we provide an adequate *n*-tuple extraction mechanism as a complementary stage to enhance further the technique when it is implemented on an elaborate processing engine such as the pyramidal *p*RAM net.

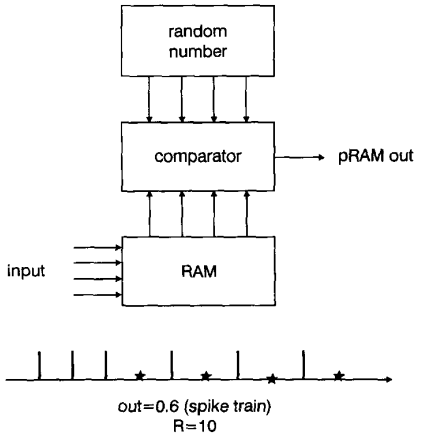

Fig. 1 *Pulse-based pRAM neuron*

1.1 Probabilistic RAM

The *p*RAM stores an *M*-bit number representing the probability α ($0 \leq \alpha \leq 1$) to fire (output 1) (Fig. 1). This is achieved by adding the stored probability to an *M*-bit random number *G*, according to eqn. 1:

$$out = \begin{cases} 1 & \text{if } \alpha + G \geq 1 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

As a result, an *n*-input *p*RAM is controlled by 2^n random variables. The conditional probability for a *p*RAM to output a 1, given a specified input vector, is given by the simplified equation [11]

$$Prob(out = 1|\mathbf{i}) = \alpha_{\mathbf{u}} \quad (2)$$

where **u** is the address obtained by the application of an *n*-input vector **i**, *out* is the *p*RAM's output and $\alpha_{\mathbf{u}}$ is the probability stored at address **u**.

IEE Proceedings online no. 19982455

Paper first received 22nd October 1997 and in revised form 23rd July 1998

M. Ouslim is with the Electronic Institute, University of Sciences and Technology, USTO – Oran, B.P. 1505 El-mnouar, Algeria

K.M. Curtis is with the Department of Electrical Engineering, University of Nottingham, Nottingham NG7 2RD, UK

When the pRAM evolves in time during R time steps, the stored probability can be estimated by the unary stochastic representation of a spike train of length R appearing at the pRAM's output. Suppose the output is observed over R time steps and there are N_1 ones; then an estimate of α_u is N_1/R (Fig. 1). The spike train generation is controlled by the mechanism used to generate G. This must provide a uniform probability distribution over all generated G values and guarantee the correct transmission of the stored probability to the input of the following pRAM. In this case, we choose $R = 2^M$, with $M = 7$ so that the memory required for the whole network will not be too excessive.

The reinforcement learning algorithm, based on a reward/punishment mechanism, is considered since it has been shown to be the most appropriate [6, 11]. The learning phase is based on the adjustment of the contents of the pRAM (αs) at each time step, according to the following rule [6]:

$$\begin{aligned}\Delta\alpha_u &= \alpha_u(t+1) - \alpha_u(t) = \Delta\alpha_u^r + \Delta\alpha_u^p \\ &= \rho\big(r(a-\alpha_u) + \lambda(\bar{a}-\alpha_u)p\big) \qquad (3)\end{aligned}$$

where $\bar{a} = 1 - a$, a is the pRAM's output, r and p are the reward and penalty signals, respectively, and $0 \le \rho \le 1$ and $0 \le \lambda \le 1$ are the learning constants.

2 pRAM multi-layer properties

As a single pyramid is used to handle a P pattern classification problem, it then becomes obvious that any node connectivity within the net must be $n \ge \log_2 P$, to allow discrimination among the P patterns at the node level. For a given image resolution, we have several alternatives to select the pRAM connectivity within different layers of the pyramid [12]. To restrict the number of possible pyramid architectures to the most relevant, the discrimination and generalisation of the pRAM net main properties are discussed here as a function of the pRAM connectivity.

2.1 Discrimination

The n-connectivity pRAM can discriminate between up to 2^n different patterns and, when taken jointly, these patterns can present differences up to $1/n$ the total input covered. However, in the case of a pyramidal net, the discrimination is limited by the pRAM with the smallest connectivity and the position of this node within the network, whether this connectivity is in the input, hidden or output layer. If n_{out} is the node connectivity in the output layer, then each input connection to the output node represents the sub-pyramid's output covering a $1/n_{out}$ fraction of the input image. That is, the net's output node may not detect the differences between patterns that occur in a $1/n_{out}$ fraction of the total input image.

It is therefore recommended to have a high connectivity at the output layer. When we move through input mappings, from the pyramid base towards its apex, the nodes map differences into similarities due mainly to the node fan-out being limited to 1. Using pRAM nodes helps to alleviate this problem since we can change the memory contents of all nodes in an incremental manner, thus forcing the output to settle at a desired value by the end of a training session.

2.2 Generalisation

In a boolean neural network, the generalisation is achieved at the network level by considering the joint contribution of several nodes within the same layer [3]. If m exemplars are used to train the net presenting n_1 nodes within its input layer, then a given pattern exemplar is divided into n_1 sub-patterns, each of which is used to train a specific node. The generalisation then covers the union of all combinations of the created n_1 subsets, each of which has at most m elements. It follows that the maximum of the generalisation ability can be quantified by $G_{max} = m^{n_1}$, where m reflects the training set diversity, and it is affected by the node connectivity at the input layer.

In the case of P patterns, an n-input node can allow, on average, $m = 2^n/P$ different representative exemplars for each pattern. Consequently, small node connectivities limit the diversity of the training set at the node level. Therefore, a relatively high value of the node connectivity is recommended, at least at the input layer when handling a P pattern recognition problem with a single pyramid. However, if we limit the pattern's diversity and distribute it over all nodes within the first layer (i.e. not localised in specific nodes), by adopting an appropriate n-tuple input mapping module based on preprocessing the training data, it might be possible that the small connectivity node generalises better when all joint node contributions are not in conflict. Accordingly, several combinations of sub-patterns seen during training can easily be created.

This discussion helped to determine the connectivity of pRAMs at the output layer, and consequently to limit the choices for the other pRAM connectivities in the input and hidden pyramid layers, according to the image resolution used.

3 Choice of n-tuple input sampling

The manner in which the n-pixels are extracted to form the n-tuple is referred to as n-tuple input sampling or mapping. These combined pixels form the sub-pattern, called the n-tuple state, which is generally used as an address to a single weightless neuron of a digital net input layer to decode the corresponding n-tuple state.

Consider two training patterns from two different categories: 0001 and 1010, and a test pattern 1011 assumed to belong to the second class. If the n-tuple size is 2 and we use the grouping 00 01 and 10 10, the test pattern 10 11 generates a score of 0 for the first category and $\frac{1}{2}$ for the second category (the score represents the number of similar n-tuples over the total number of n-tuples). However, if the grouping is such that the first and the third bits form one n-tuple and the second and the fourth form the second n-tuple, then we get the states 00 01 for pattern 1 and 11 00 for pattern 2, and the test pattern presents 11 01, which leads to a score of $\frac{1}{2}$ for both categories. As a result, the scheme fails in classifying the test pattern. It becomes clear that when a grouping is chosen, boolean functions for each n-tuple are generated according to the states present in the training set of that particular n-tuple. If the grouping changes, the expressions of these boolean functions automatically change.

In this study, the order of the n-tuple elements is irrelevant in defining the n-tuple; what matters the most is the significance of the n-tuple elements, as we are not interested in specific n-tuple states favoured by a specific problem, but rather in extracting all n-tuple states. Under this condition, the total number of all possible mappings tot_{map} which sample n-pixels (n-tuple), from an image of Qn pixels, into a set of Q

strings of length n, is given by $(C_n^Q = Q!/(Q-n)!n! = (Q(Q-1) \ldots (Q-n+1))/n!)$

$$tot_{map} = C_n^{Qn} C_n^{Qn-n} C_n^{Qn-2n} \ldots C_n^{Qn-in} \ldots C_n^{Qn-(Q-1)n} = \frac{(Qn)!}{(n!)^Q} \quad (4)$$

Searching all this space for an optimum solution is not computationally possible. For example, in the case of $Qn = 8 \times 8$ and $n = 4$, we get $Tot_{map} = 1.05 \times 10^{67}$. This is why we propose an input mapping based on data analysis; we believe that extracting knowledge from the actual data to select the n-tuples is the most appropriate way to exploit the n-tuple technique to its fullest.

3.1 Proposed n-tuple sampling scheme

In a P pattern recognition problem, the n-tuples must fulfil the following conditions:

(i) the n-tuples must convey the similarities between patterns in the same class.

(ii) the n-tuples must convey the differences between patterns in different classes.

One way to achieve these objectives is to take into consideration the local variations seen by each pixel within the image, according to the data used. As a result, we get L (image size in pixels) functions, each of which can be represented by P (number of patterns) strings of length T (size of each training set). Since each individual pixel goes through a cycle of variations, we suggest to compute the resultant probability of a given n-tuple to be in a given state, based on the individual probability densities of the constituent pixels. It is therefore necessary to assume that the pixels which form the n-tuple are not correlated. This simplification is done to avoid excessive computation at this preliminary stage. Hence, we use the frequency of occurrence of a pixel in the state 1 as the main metric to represent changes at the pixel level while handling one training set at a time. Normally, if an n-tuple state is characteristic for a given training set, this state should be approximately constant (i.e. occurs with a high probability) and should have a low probability of occurrence for other training sets. Hence, to group pixels in n-tuples we must test for their frequency of occurrence in a particular state.

To mathematically formulate this concept, we restrict the n-tuple size and the number of patterns to be classified to 4. In this case, an n-tuple element w is characterised by four frequencies $\{f_{w0}, f_{w1}, f_{w2}, f_{w3}\}$ recorded for the four classes used. Given the threshold value ε (normally less than 0.5), we search for n-tuple elements that respond to the following criteria:

(i) $f_{wi} < \varepsilon$ or $f_{wi} > 1 - \varepsilon$ for all $i \in \{0, 1, 2, 3\}$

(ii) $f_{wi} < \varepsilon$ and $f_{wj} > 1 - \varepsilon$ for all $i, j \in \{0, 1,2, 3\}$ and $i \neq j$

Since n pixels are to be grouped in the same n-tuple, we have a multitude of choices to fulfil these criteria.

In the case of a black and white image, the pattern space can be at most divided into two partitions (one dichotomy) at the pixel level. In this context, we form four main partitions to group all image pixels. Partition 1 contains pixels with similar frequencies for all patterns used, represented by P similar patterns and one characteristic state. Partition 2 gathers pixels presenting exactly $P/2$ similar frequencies, represented by $P/2$ similar patterns and $C_P^P/2$ (i.e. $= p!/((p - p/2)! \times (p/2)!)$) characteristic states. Partition 3 contains pixels with $P - 1$ similar frequencies, represented by $P - 1$ similar patterns and P characteristic states. Partition 4 groups all remaining pixels not included in the previous three partitions.

For clarity, we use the following specific example. Let $\varepsilon = 0.2$. If the pixel w presents the frequencies 0.4 0.7 0.6 0.2, then w has an indeterminate state and belongs to partition 4. However, if the frequencies are 0.9 0.9 0.1 0.9, w has the state: (1,1,0,1) and belongs to Partition 3.

An n-tuple is desirable that incorporates pixels with the corresponding states (0,0,0,0) partition 1, (0,1,0,1) partition 2, (1,0,0,1) partition 2 and (0,0,0,1) partition 3. This is because the n-tuple states are (0,0,1,0) for pattern 1, (0,1,0,0) for pattern 2, (0,0,0,0) for pattern 3, and (0,1,1,1) for the fourth pattern, achieving discrimination among the four patterns at the n-tuple level. This n-tuple sampling scheme was set up as an algorithm with the following steps:

(i) Compute the frequency of occurrences of image pixels to be *on* for all P training sets used.

(ii) Group pixels into four main partitions, according to their ability to discriminate between the pattern classes.

(iii) Transform frequencies into pixel states (threshold levels ε can be varied according to needs).

(iv) Form n-tuples by taking pixels from partition 1 and partition 2. Conditions are set up to randomly search for pixels from partition 2 so as to allow discrimination among the four patterns. Pixels from partition 1 and partition 3 are then grouped. When partition 1 is exhausted, the process continues for the remaining partitions.

Note that partition 4 contains pixels which present an indeterminate state when thresholding their frequency of occurrences. In the extreme case, a pixel with $f = 0.5$ does not provide any contribution to the particularities of the pattern (50% of the training exemplars have this pixel with the state 0 and the remaining 50% represent it with the state 1). It seems that to enhance the generalisation property, it is beneficial to pack this type of pixel into the same n-tuple. However, searching for such pixels to be part of another partition complicates the algorithm and slows down its operating speed.

Note that this particular partitioning of pixels into four partitions is mainly because we want to provide a simple algorithm that can operate as fast as possible. This algorithm is general in its application, since it does not assume any underlying specific knowledge about the pattern classes handled.

However, the distribution of the n-tuples at the pyramidal net input is pivotal to the successful application of the pRAM net. Indeed, if the connectivity of the output layer is n_{ho}, then the image is divided into n_{ho} equal sized quadrants, each of which is independently handled by the use of sub-pyramids built at precedent layers. Therefore, we can evenly arrange the obtained n-tuple set through all these parts, so that each sub-pyramid can contribute to the decision made at the output. As a result, we get a two-stage n-tuple input mapping. In the first stage, we compact pixels conveying information about the object within appropriate n-tuples, to increase discrimination between different patterns and decrease it between noisy patterns.

In the second stage, we evenly distribute the obtained n-tuples through the pyramid base. For this purpose, we define the joint n-tuple distance (JND) as the number of n-tuples which represent similar states when P, $P-1$, .., 2 patterns are jointly taken using the same n-tuple input mapping. By gathering all joint n-tuple distances for various input mappings covering all n-tuples, we obtain the joint n-tuple distance map, where an n-tuple can be one of P different types corresponding to P, $P-1$, ..., 2, 0 patterns, respectively, with the same state for that particular n-tuple. This map gives the information about the correlation among the patterns, as well as the spatial distribution of all n-tuples.

Note that this representation is obtained by recording the frequency of occurrence of all n-tuple states for all n-tuples using the transformed training sets, according to the n-tuple input mapping used. This allows us to define the characteristic image that reflects a high similarity among all patterns within a given training set, and it is obtained by selecting the n-tuple attractor state (i.e. the most frequently occurring state) for all n-tuples. By a simple comparison between all characteristic images, we then get the JND that represents the correlation among all patterns used.

4 Results and comments

Several computer simulations were carried out to highlight the effect of using different node connectivities and to test the n-tuple input mappings on a pyramidal neural network [12].

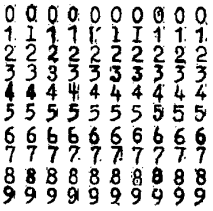

Fig. 2 *Sample of 10 digits used*

For this purpose, real-life data were handled, consisting of the digits shown in Fig. 2. The size of the image at 24 × 16 suggests the use of various pRAM net topologies. In the case of the four-pattern recognition problem, four digits among the 10 available (0, 6, 8, 9) were selected as being one of the most correlated cases. Each digit was represented by 150 training exemplars and 150 test exemplars. A 4–4–4–6 pRAM net was simulated to perform the recognition task. This choice was mainly dictated by the saving in storage so that the simulation could run on a simple IBM PC.

Three n-tuple input mappings were tested: the structured (i.e. regularly samples n consecutive pixels),the data derived using the proposed algorithm, and a permuted version of the proposed algorithm. The spatial distributions of the n-tuples represented by these mappings are indicated using the JND distance map given in Fig. 3. Table 1 summarises the proportion of each n-tuple type obtained from a corresponding n-tuple input mapping. The net's performance is measured using the convergence and recognition rates, as these are the most common metrics used for pattern recognition problems. The obtained results are illustrated in Fig. 4 and Table 2. The learning constants ρ and λ were set up as 0.8 and 0.008, respectively.

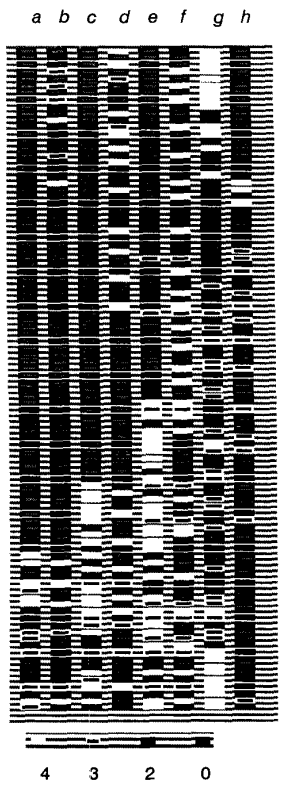

Fig. 3 *Joint n-tuple distance map for digits*
$P = 4$ and $P = 10$
Case of 24 × 16 digits; each entry represents an n-tuple.
Input mappings used are listed in order of appearance: (4,5,6,7) digits: a = mapped, b = permuted, (0,1,2,3) digits: c = mapped, d = permuted, (0,6,8,9) digits: e = mapped, f = permuted, g = linear 10 digits: h = derived from b–d–f plotted for (3,4,5,9)

Table 1: Proportion of each n-tuple type obtained from corresponding n-tuple input mapping

n-tuple mapping	Group of digits	n-tuple types			
		0	2	3	4
a, b	4,5,6,7	65	23	6	6
c, d	0,1,2,3	56	20	5	19
e, f	0,6,8,9	35	27	11	27
h	10-digits	26	58	13	3

Proportion of n-tuples is given as a percentage of a total of 96

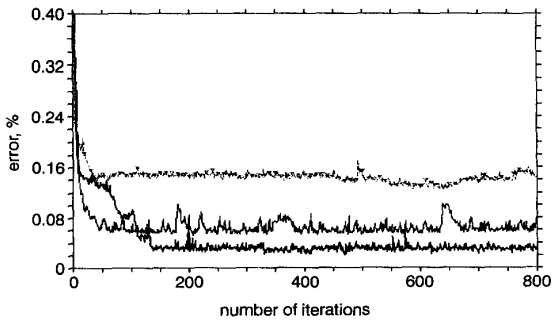

Fig. 4 *pRAM net convergence rate*
Various n-tuple mappings case of 4 digits
——— permuted
▽– – –▽ structured linear
——— data-mapped

Table 2: Recognition rate for 4 digits (0,6,8,9)

n-tuple input mapping	Training set recognition (%)	Test set recognition	Convergence time (IBM PC 66MHz)
Data-mapped	99	80	9343s (260 iterations)
Structured (linear)	97	87	4297s (100 iterations)
Permuted	100	93	6200s (140 iterations)

Fig. 4 shows that the network's convergence is better in the case of the permuted and data-based mappings than for the linear mapping. Using the permuted mapping, we managed to achieve a low convergence error rate approximated at 2%, compared to 13% for the structured mapping and 6% for the data-based mapping without permutation. We noted that the net based on the structured mapping needed fewer iterations to converge, where the estimated convergence time is 4297 s compared to the other nets, as illustrated in Table 2. This is due to the high value of the convergence error recorded for the structured *n*-tuple mapping. Fig. 4 presents a stable error for the structured mapping compared to the data-based and permuted mappings. These fluctuations are the result of changes in the *p*RAM memory contents, which vary in the range [0,1].

However, Table 2 illustrates that the network using the permuted mapping achieves the highest recognition rate of 93% (i.e. the mean value of the recognition rates corresponding to the four digits), whereas the non-permuted mapping only achieved 80%, even worse than that of the structured input mapping. This result can be justified by consulting the JND map. Indeed, we see that the structured mapping presents relatively good distribution of all *n*-tuple types throughout the input layer of the pyramidal net compared to the data-based input mapping, as illustrated by their respective JNDs *g* and *e* in Fig. 3. In rearranging the obtained *n*-tuples, using the data-based algorithm, we obtain the JND *f*. The resultant *n*-tuple mapping, called permuted mapping, presents better spatial distribution of the *n*-tuples than the original data-based mapping; this is shown by comparing the two JNDs *e* and *f* in Fig. 3.

This result allows us to state that breeding a good *n*-tuple input mapping is not sufficient when used with a *p*RAM pyramidal network; in addition, we must provide a uniform distribution of the *n*-tuple types at the network input.

The *p*RAM network was extended to tackle the ten-digit recognition problem. This necessitates increasing the output layer to four nodes. The output was appropriately coded to represent each pattern uniquely. The codes are chosen so that each output node has the same possibility to be either one or zero. Hence, we used the following codes: (1100,0), (0001,1), (1010,2), (0011,3), (0100,4), (0101,5), (0110,6), (0111,7), (1000,8) and (1001,9). To get the JND of ten patterns, it is necessary to perform exhaustive comparisons between all representative patterns. In our case to avoid such computations, we construct the resultant JND from three different JNDs corresponding to the three arbitrarily chosen groups of four digits (0,1,2,3), (4,5,6,7) and (0,6,8,9), respectively. We then distributed the *n*-tuples uniformly to obtain a balanced set of *n*-tuple types at the input of the *p*RAM net. These permuted JNDs are used to construct the JND for the ten digits, by taking *n*-tuples from each JND without any overlap. The obtained JND represents a particular *n*-tuple input mapping, and it was used to generate the transformed training and test sets.

To evaluate the effect of this input mapping on the *p*RAM net when it performs the recognition of ten digits, we conducted several tests covering three *p*RAM networks; the 4–4–4–6, the 8–6–8 and the 6–8–8. The first configuration saves memory but slows down the training session as the pyramid depth is deeper; even worse, it may happen that this net does not converge due to the diversity of the training set, since the connectivity used allows less than two different exemplars per pattern. The second configuration 6–8–8 limits the number of pyramid layers at the cost of a higher node connectivity requiring 7168 memory words, compared with 13696 in the case of the 8–6–8 net. In addition, these configurations present a high connectivity at the output layer, as recommended in the discussion presented in Section 2.

We used the structured mapping for the three named nets, and we trained them using the learning constants $\rho = 0.95$ and $\lambda = 0.01$. The training set covered 1500 patterns, representing 150 examples for each digit. We noted that, in the case of the structured linear input mapping, the 6–8–8 net outperforms the others concerning the convergence error, which is around 12% for the 6–8–8, 15% for the 8–6–8 net and 30% for the 4–4–4–6 network. These networks have been trained for 1000 iterations for the 4–4–4–6 network and 500 iterations for the 6–8–8 and the 8–6–8 networks. The 4–4–4–6 network configuration did not show any improvement in the convergence error rate; it was then judged unable to learn this problem.

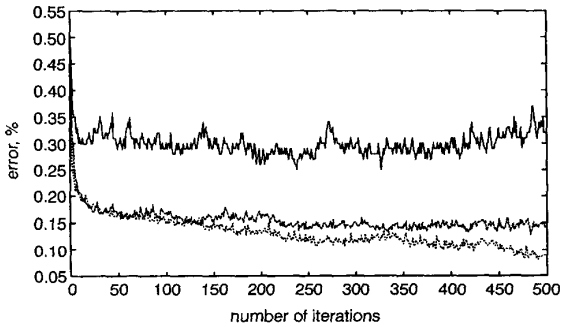

Fig. 5 *Convergence error for 10 digits (permuted mapping)*
——— net4446
– – – – net868
····· net688

Table 3: Recognition rate for 10 digits

n-tuple input mapping	*p*RAM net architectures			
	8–6–8		6–8–8	
	Training set	Test set	Training set	Test set
Linear	97	84	96	83
Permuted	96	86	98	89

When the permuted version of the data-based mapping was used, the error dropped to 9% for the 6–8–8 network compared to 14% for the 8–6–8 network. This was not the case for the 4–4–4–6 network, where the n-tuple input mapping did not enhance the network convergence, as indicated in Fig. 5. However, the results summarised in Table 3 correspond to the 6–8–8 and the 8–6–8 nets, confirming that the permuted mapping outperforms the linear mapping, with the 6–8–8 net performing best. This is an interesting result, as we get better performance with less memory capacity requirement in using the 6–8–8 net compared to the 8–6–8 net.

As far as the training time is concerned, the 8–6–8 net took 360 iterations to converge to an acceptable low error (14%), corresponding to 3h on a SUN workstation; compared to 218 iterations, corresponding to 1h 47min, for the 6–8–8 net. These are relatively slow training sessions, mainly due to the use of the single pyramid where the number of patterns used is relatively high (1500).

However, in taking into consideration the recognition rate metric, even though we verified that the permuted input mapping performs better than the structured mapping, we achieved an overall result of 89% recognition success. This is compared to other studies where more than 90% of the recognition success rate is generally reported for the recognition of handwritten digits [13].

As the permuted n-tuple input mapping was based on the JND (derived in this case in an indirect way), better results are expected for a judiciously selected JND for the ten-pattern recognition problem. The use of more than one pyramid, to allow for the overlap among the n-tuples, plays a positive role in enhancing the performance. These suggestions are left as a possible continuation to this work.

However, we should bear in mind that this application is meant to be a tool to verify the adequacy of the proposed approach to select adequate n-tuples and the effect of distributing them at the input of a Boolean neural network. We did not intend to solve specifically the digits recognition problem: this is why no complete comparative study was undertaken with other techniques. Nevertheless, the good results obtained are very motivating and suggest that the use of such a scheme should be further investigated.

5 Conclusions

In this paper, pRAM nets have been simulated and used to tackle a P image recognition problem. This study has emphasised the benefit of using a pRAM net with different node connectivities from one layer to the other, as each connectivity affects the main properties of the net in a conflicting manner. By using such topologies, we can guarantee a good balance between the net's main properties; generalisation and discrimination.

A two stage n-tuple input mapping based on data analysis was proposed. The first stage extracts n-tuples according to discrimination among patterns at the n-tuple level. This delivers a mixture of n-tuple types, and when used with a single pyramidal pRAM net without any overlap among n-tuples, required an adequate distribution of the obtained n-tuples at the pyramid base. In this context, the joint n-tuple distance (JND) was introduced and used to derive the final n-tuple input mapping. The application of these propositions to digit recognition was successful and confirmed their effectiveness when handling a P pattern recognition task.

We should note, however, that the complexity increases for big values of P (the number of patterns used) when extracting the JND. This limitation of the n-tuple input mapping can be avoided by using an indirect way to derive the JND, as in the case for the digit recognition problem proposed in this study.

We also believe that extending the variability of the node fan-in within the same layer makes the network perform better.

6 References

1 BLEDSOE, W.W., and BROWNING, I.: 'Pattern recognition and reading by machine'. Proceedings of the Eastern Joint Computer conference, 1959, pp. 225–233

2 ALEKSANDER, I., and STONHAM, T.J.: 'Guide to pattern recognition using random-access memories', *IEE J. Comput. Digit. Tech.*, 1979, **2**, (1), pp. 29–40

3 ALEKSANDER, I.: 'Weightless neural models for cognitive design', *J. Intell. Syst.*, 1992, **2**, pp. 31–52

4 OUSLIM, M., and CURTIS, K.M.: 'Automatic visual inspection based upon a variant of the n-tuple technique', *IEE Proc. Vis. Image, Signal Process.*, 1996, **143**, (5), pp. 301–309

5 FILHO, E.C., FAIRHURST, M.C., and BISSET, D.L.: 'Adaptive pattern recognition using goal seeking neurons', *Patt. Recog. Lett.*, 1991, **12**, pp. 131–138

6 CLARKSON, T.G., GORSE, D., TAYLOR, J.G., NG, C.K., , and : 'Learning probabilistic RAM nets using VLSI structures', *IEEE Trans. Computers*, 1992, **41**, (12), pp. 1552–1561

7 AL-ALAWI, R., and STONHAM, T.J.: 'Functionality of a multi-layer Boolean neural network', *Electron. Lett.*, 1989, **25**, (10), pp. 658–659

8 DE CARVALHO, A., FAIRHURST, M.C., and BISSET, D.L.: 'An integrated Boolean neural network for pattern classification', *Patt. Recog. Lett.*, 1994, **15**, pp. 807–813

9 ROHWER, R., and MORCINIEC, M.: 'Theoretical and experimental status of the n-tuple classifier'. Report, NCRG/4347, Aston University, Birmingham, UK, May 1995

10 STONHAM, T.J.: 'Improved hamming-distance analysis for digital learning networks', *Electron. Lett.*, 1977, **13**, (6), pp. 155–156

11 CLARKSON, T.C.: 'The pRAM as a hardware realisable neuron'. Proceedings of Neural Networks, 1992, pp. 140–146

12 OUSLIM, M.: 'Analysis of the n-tuple technique based upon two perspectives to determine n-tuple groupings'. PhD Thesis, University of Nottingham, 1997

13 GADER, P.D., and KHABOU, M.A.: 'Automatic feature generation for handwritten digit recognition', *IEEE Trans. Pattern Anal. Mach. Intell.*, 1996, **18**, (12), pp. 1256–1261