A Binary Correlation Matrix Memory *k*-NN Classifier with Hardware Implementation

Ping Zhou, Jim Austin and John Kennedy
Computer Science, University of York, York YO10 5DD, UK
[Zhoup|austin]@cs.york.ac.uk

Abstract

This paper describes a generic and fast classifier that uses a binary CMM (Correlation Matrix Memory) neural network for storing and matching a large amount of patterns efficiently, and a *k*-NN rule for classification. To meet CMM input requirements, a robust encoding method is proposed to convert numerical inputs into binary ones with the maximally achievable uniformity. To reduce the execution bottleneck, a hardware implementation of the CMM is described, which shows the network with on-board training and testing operates at over 200 times the speed of a current mid-range workstation, and is scaleable to very large problems. The CMM classifier has been tested on several benchmarks and, comparing with a simple *k*-NN classifier, it gave less than 1% lower accuracy and over 4 and 12 times speed-ups in software and hardware respectively.

1 Introduction

Desirable characteristics of Correlation Matrix Memory (CMM) neural networks include simple and quick training, and highly flexible and fast search ability [1]. Whereas most neural networks need a long iterative training times, a CMM is trained using an one-shot storage mechanism and simple binary operations. The CMM has been used as a match engine in a number of successful applications, e.g. symbolic reasoning in the AURA (Advanced Uncertain Reasoning Architecture) approach [2], chemical structure match [4] and post code matching. This work investigates its use for pattern classification tasks. It is known that the *k*-NN rule [5] is applicable to a wide range of classification problems. However, this method is too slow to use for many applications with large amounts of data. To speed up, previous researchers have considered reducing training data [6] and improving computational efficiency via complex pre-processing of training data [7]. In contrast to these, a CMM is a simple, general and powerful approach which can be used to store a large number of training patterns efficiently, and to retrieve both exact and near matches quickly for a test pattern. Therefore, the combination of CMM and *k*-NN techniques may result in a generic and fast classifier.

For most classification problems, patterns are in the form of multi-dimensional real numbers, and appropriate quantisation and encoding are needed to convert them into binary inputs to a CMM. A robust quantisation and encoding method is developed to meet requirements for CMM input codes, such as uniformity, orthogonality and

BMVC 1998 doi:10.5244/C.12.22

sparseness [3], and to overcome the common problem of identical data points in many applications, e.g. background of images or normal features in a diagnostic problem.

The execution of the CMM was quickly identified as the bottle neck in the processing by an analysis of the AURA [2] method. To reduce this bottleneck, the CMM has been implemented in dedicated hardware, that is the PRESENCE architecture. The primary aim is to improve the execution speed over conventional workstations in a cost effective way. This work was also motivated by the needs of many research projects applying the CMM to commercial problems mentioned above.

The next section discusses the CMM for pattern classification and the robust uniform (RU) encoding method, followed by descriptions of the PRESENCE architecture (the hardware implementation of the CMM). Experimental results are presented in Section 4, and concluding remarks in the last section.

2 CMM for Pattern Classification

Figure 1 shows the architecture of the CMM classifier. The RU encoder (as detailed in 2.2) quantises numerical inputs and generates binary codes; the CMM engine stores training patterns and matches stored patterns close to a test pattern to supply to a conventional *k*-NN module for classification. Both the CMM and *k*-NN modules are needed as the CMM is fast but produces spurious errors as a side effect [3]. These are removed through the application of the *k*-NN rule. More specifically, the speed of the classifier benefits from the use of the CMM for fast training and matching to pre-select a sub-set patterns from a large amount of training data; the accuracy gains from the application of the *k*-NN rule to the sub-set in the original space to reduce information loss and noise in the encoding and match processes.

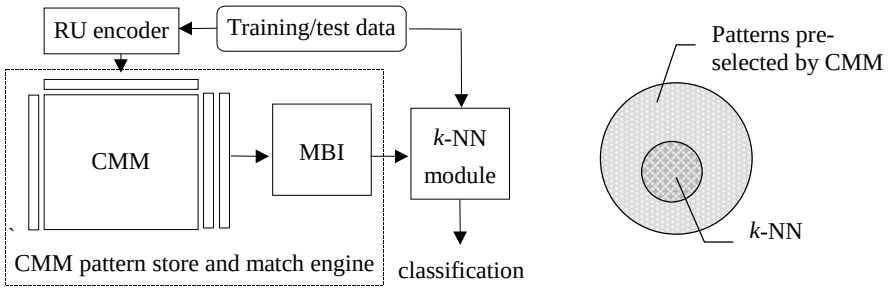


Figure 1: Architecture of the CMM classifier

2.1 Pattern Match and Classification with CMM

In the CMM there is a binary matrix $\boldsymbol{M}$ and, prior to any learning, all of its elements are set to '0'. In a training process a unique binary vector (or separator as often called) $\boldsymbol{s}_i$ is generated to label an unseen input binary vector $\boldsymbol{p}_i$; the CMM learns through the association of the two vectors by performing the following logical ORing operation,

$$\boldsymbol{M} = \bigvee_i \boldsymbol{s}_i^T \boldsymbol{p}_i \qquad (1)$$

In a recall process, for a given test input vector $\boldsymbol{p}_k$, the CMM performs,

$$\boldsymbol{v}_k = \boldsymbol{M}\boldsymbol{p}_k^T = \bigvee_i \boldsymbol{s}_i^T \boldsymbol{p}_i \boldsymbol{p}_k^T \quad (2)$$

followed by thresholding $\boldsymbol{v}_k$ and recovering individual separators using a MBI (Middle Bit Index) method [2]. For speed, it is appropriate to use a fixed thresholding method and the threshold is set to the level equal to the number of '1' bits in the input pattern to allow exact match, or a low level to match a proportion of the input pattern as detailed below.

To understand the recall properties of the CMM, consider the case where a known pattern $\boldsymbol{p}_k$ is represented, then Equation 2 can be written as,

$$\boldsymbol{v}_k = \boldsymbol{s}_k^T \boldsymbol{p}_k \boldsymbol{p}_k^T + \bigvee_{i \neq k} \boldsymbol{s}_i^T \boldsymbol{p}_i \boldsymbol{p}_k^T = n_p \boldsymbol{s}_k^T + \bigvee_{i \neq k} \boldsymbol{s}_i^T \left(\boldsymbol{p}_i \boldsymbol{p}_k^T \right) \quad (3)$$

where $n_p = \boldsymbol{p}_k \boldsymbol{p}_k^T$ is a scalar, i.e. the number of '1' bits in $\boldsymbol{p}_k$. When two different patterns are orthogonal to each other, the inner product $\boldsymbol{p}_i \boldsymbol{p}_k^T = \boldsymbol{0}$ for $i \neq k$ and the second term in the above disappears. Hence a perfect recall of $\boldsymbol{s}_k$ can be obtained by thresholding $\boldsymbol{v}_k$ at the level n_p. In practice 'partially orthogonal' codes may be used to increase the storage capacity of the CMM. In such a situation $\boldsymbol{v}_k$ contains both the correct recall (the first term) and incorrect ones (the second term) as the results of $\boldsymbol{p}_i \boldsymbol{p}_k^T \neq \boldsymbol{0}$ for $i \neq k$. It is possible to remove the noise via appropriately thresholding $\boldsymbol{v}_k$ (as $\boldsymbol{p}_i \boldsymbol{p}_k^T \leq n_p$ for $i \neq k$) and post-processing (e.g. applying the *k*-NN rule). Sparse codes are usually used, i.e. only a few bits in separators and input vectors being set to '1', as this maximises the number of codes and minimises the computation time [3]. These requirements for input codes are often met by an encoder as detailed below.

The CMM exhibits an interesting 'partial match' property when the data dimensionality d is larger than one and input vector $\boldsymbol{p}_i$ consists of d concatenated components. We have $\boldsymbol{p}_i \boldsymbol{p}_k^T \neq 0$ for $i \neq k$ in Equation 3 if the two different patterns $\boldsymbol{p}_i$ and $\boldsymbol{p}_k$ have some common components. Therefore, $\boldsymbol{v}_k$ also contains separators for partially matched patterns, and these separators can be obtained at lower threshold levels. This partial or near match property is useful for pattern classification as it allows the retrieval of stored patterns which are close to the test pattern in Hamming distance.

From those training patterns matched by the CMM engine, a test pattern is classified using the *k*-NN rule. Distances are computed in the original input space to minimise the information loss due to quantisation and noise in the above match process. As the number of matches returned by the CMM is much smaller than the number of training data, the distance computation and comparison are dramatically reduced compared with the simple *k*-NN method. Therefore, since the CMM stage is very fast, the CMM based *k*-NN classifier can be faster.

2.2 Robust Uniform Encoding

In addition to the above sparseness and orthogonality, another primary requirement for CMM input codes is that they should be distributed as uniformly as possible in order to avoid some parts of the CMM being used heavily while others are rarely used. Figure 2 shows three stages of the encoding process, that is quantising d-dimensional real numbers, x_i, generating sparse and orthogonal binary vectors, c_i, and concatenating

them to form a CMM input vector.

The code uniformity is met at the quantisation stage. For a given set of N training samples in some dimension (or axis), it is required to divide the axis into N_b small intervals, called bins, such that they contain uniform numbers of data points. As the data often have a non-uniform distribution, the sizes of these bins should be different. It is also quite common for real world problems that many data points are identical. For instance, there are 11%-99.9% identical data in benchmarks used in this work. Our robust quantisation (RQ) method described below is designed to cope with the above problems and to achieve a maximal uniformity.

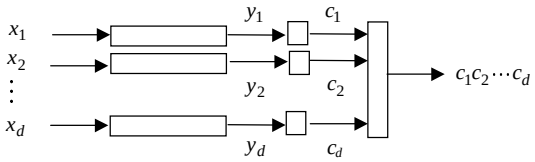


Figure 2: Quantisation, code generation and concatenation

In our method data points are first sorted in ascending order, N_i identical points are then identified, and the number of non-identical data points in each bin is estimated as $N_p = (N - N_i)/N_b$. Bin boundaries or partitions are determined as follows. The right boundary of a bin is initially set to the next N_p-th data point in the ordered data sequence; the number of identical points on both sides of the boundary is identified; these are either included in the current or next bin. If the number of non-identical data points in the last bin is N_l and $N_l \geq (N_p + N_b)$, N_p may be increased by $(N_l - N_p)/N_b$ and the above partition process may be repeated to increase the uniformity. Boundaries of bins obtained become parameters of the encoder in Figure 2.

Sizes of bins (or the number of bins) determine the match 'neighbourhood' since samples falling in the same bin have the same quantised value. In an extreme case when $N_b = 1$, the CMM matches all training samples for any test data and our complete system becomes a standard *k*-NN classifier. In general it is appropriate to chose N_b such that each bin contains a number of samples, which is larger than *k* nearest neighbours for the optimal classification.

3 The PRESENCE Architecture

3.1 Architecture design

Some of important design decisions for implementing the CMM were: the system should use cheap memory, and should not attempt to embed both the weight storage and the training and testing in hardware (VLSI). This arises because the applications commonly use CMMs with over 100Mb of weight memory, which would be difficult and expensive to implement in custom silicon. The system must be hosted on industry standard buses to allow widespread application, thus VME and PCI were chosen.

The PRESENCE architecture implements the control logic and accumulators necessary to implement the core of the CMM. As shown in Figure 3a the CMM takes a

set of binary inputs within the input pattern. Each input selects rows from the CMM that will be added into the accumulators (note that the input x the weights operation is implicit in this process). The accumulated data is then thresholded using *L*-max [8] or fixed global thresholding. Finally, the data is then returned to the host for further processing. The outline of the PRESENCE architecture is shown in Figure 3b. The architecture consists of a bus interface, a buffer memory which allows interleaving of memory transfer and operation of the PRESENCE system, a SATCON and SATSUM combination that accumulates and thresholds the weights. The data bus connects to a pair of memory spaces, each of which contains a control block, an input block and an output block. Thus the PRESENCE card is a memory mapped device, that uses interrupts to confirm the completion of each operation. To maintain an efficient use of input memory the bits that are set to one in the input, ***p***, are passed to the processor card as 'index values' one for each bit set. For efficiency, two memory input/output areas are provided so that one can be acted on from the external bus while the other is used by the card. The control memory input block feeds to the control unit, which is a FPGA device programmed to carry out all necessary operations. The input data (index values) are fed to the weights on the card and the area of memory that is read is then passed to a block of accumulators. In our current implementation the data width of each FPGA device is 32 bits, which allows us to add a 32 bit row from the weights memory in one cycle per device.

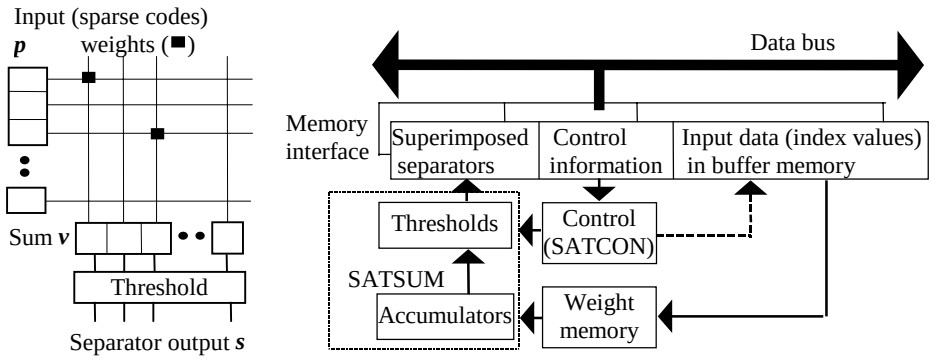


Figure 3: (a) correlation matrix memory, and (b) overall architecture of PRESENCE

Currently we have 16Mb of 20ns static memory implemented on the VME card, and 128 Mb of dynamic (60ns) memory on the PCI card. The accumulators are implemented along with the thresholding logic on another FPGA device (SATSUM).

To enable the SATSUM processors to operate faster, a 5 stage pipeline architecture was used. The stages of which are; index value count: latch address into buffer memory: add index to memory offset: latch result of index calculation: access the weights memory with the address. The use of this pipeline reduces the data accumulation time from 175ns to 50ns. All PRESENCE operations are supported by a C++ library that is used in all AURA applications.

The design of the SATCON allows many SATSUM devices to be used in parallel in a SIMD configuration. The VME implementation uses 4 devices per board giving a 128 bit wide data path. In addition the PCI version allows daisy chaining of cards allowing a 4 card set for a 512 bit wide data path.

The complete VME card assembly is shown in Figure 4. The SATCON and SATSUM devices are mounted on a daughter board for simple upgrading and alteration. The weights memory, buffer memory and VME interface are held on the mother board.

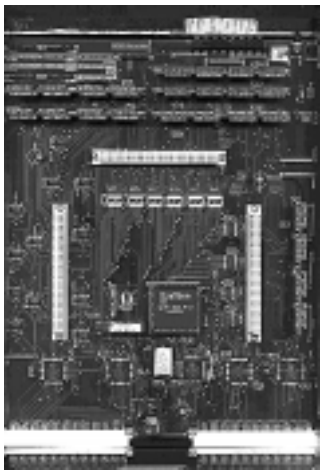
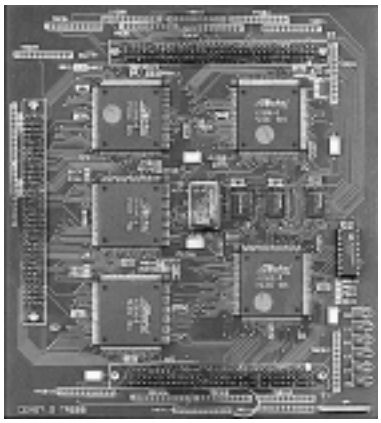
(a)   (b)

Figure 4: The VME based PRESENCE card (a) motherboard, and (b) daughterboard

3.2 Performance

By an analysis of the state machines used in the SATCON device the time complexity of the approach can be calculated. Equation 4 is used to calculate the processing time, *T*, in seconds to recall the data with *N* index values, a separator size of *S*, *R* 32 bit SATSUM devices, and the clock period of *C*.

$$T = C\left[23 + \left((s-1)/32R + 1\right)\left(N + 38 + 2R\right)\right] \quad (4)$$

A comparison with a Silicon Graphics 133MHz R4600SC Indy is given in Table 1. This shows the speed up of the matrix operation (Equation 2) for our VME implementation (128 bits wide). The timings are for a fixed threshold. The values for processing rate are given in millions of binary weight additions per-second (MW/s). In this implementation the system cycle time needed to sum a row of weights into the counters (i.e. time to accumulate one line) is 50ns for the VME version and 100ns for the PCI version. In the PCI form, we will use 4 closely coupled cards, which result in a speed-up of 432.

Platform	Processing Rate	Relative Speed
Workstation	11.8 MW/s	1
1 Card VME implementation	2557MW/s	216
Four card PCI system (estimate)	17,114MW/s	432

Table 1: Relative speed-up of the PRESENCE architecture.

The build cost of the VME card was half the cost of the baseline SGI indy machine given above, when using 4Mb of 20ns static RAM. In the PCI version the cost is greatly

reduced through the use of dynamic RAM devices allowing a 128Mb memory to be used for the same cost, allowing only a 2x slower system with 32x as much memory per card (note that 4 cards used in table 1 hold 512Mb of memory).

The training and recognition speed of the system are approximately equal. This is particularly useful in on-line applications, where the system must learn to solve the problem incrementally as it is presented. In particular, the use of the system for high speed reasoning allows the rules in the system to be altered without the long training times of other systems. Furthermore our use of the system for a *k*-NN classifier also allows high speed operation compared with a conventional implementation of the classifier, while still allowing very fast training times.

To appreciate the utility of our implementation consider its use as a pattern recognition system for a small mobile robot that must follow a pre-planned route. The aim is for the robot to follow the route using the images it captures on a previous guided tour of the route. To do this we use the N tuple pre-processing method [9]. This method takes an image frame and performs simple feature analysis on the image which is passed to the CMM as a vector containing a fixed number of bits set to one, each bit represents a feature that it has found in the image. Consider a camera, taking images at 25 frames per second, at a resolution of 512^2. If the image is sampled at 10% with an N tuple size of 4, then 6553 features ('tuples' in N tuple terminology) will be sampled and passed to the CMM in a vector of size 104856 bits. If a separator is used per image frame, and each separator is unique and has 2 bits set, then a separator of size 10240 will allow 20480 separators to be stored using a memory of size 128Mb (the PCI memory size). At 25 frames per second, this allows the robot to store images for 13 minutes. Recognition of the frames can be also be performed at frame rate. Using the 4 card PCI implementation, almost an hour of frame rate video can be stored and recognised. For guidance, the robot stores the direction along with each video image of the scene ahead as it is hand guided through the environment. In recognition, the recogniser finds the image that best matches the current view and recalls the guidance information. This shows the potential use of the technology in a novel application which is difficult to achieve in a cost effective way by any other method.

4 Results on Benchmarks

Performance of the robust quantisation method and the CMM classifier have been evaluated on four benchmarks consisting of large sets of real world problems from the Statlog project [10], including a satellite image database, letter image recognition database, shuttle data set and image Segmentation data set.

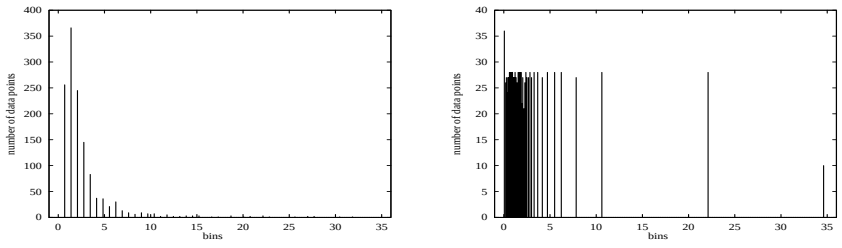


Figure 5: Distributions of the image segment data for (a) equal bins, (b) RQ bins

To visualise the result of quantisation, Figure 5a shows the distribution of numbers of data points of the 8th feature of the image segment data for equal-size bins. The distribution represents the inherent characteristics of the data. Figure 5b shows our robust quantisation (RQ) has resulted in the uniform distribution desired.

We compared the CMM classifier with the simple *k*-NN method, multi-layer perceptron (MLP) and radial basis function (RBF) networks [11]. The performance of interest are classification rate (c-rate) on test data sets and relative speed (r-speed). In the evaluation we used the CMM software libraries developed in the project AURA at the University of York. It is appropriate to set 1-3 '1' bits in input vectors and separators. Experiments were conducted to study influences of a CMM's size on c-rate and r-speed measured against the *k*-NN method (as shown in Figure 6), where the r-speed of the CMM classifier includes the encoding, training and test time. The effects of the number of bins N_b on the performance were also studied (Figure 7).

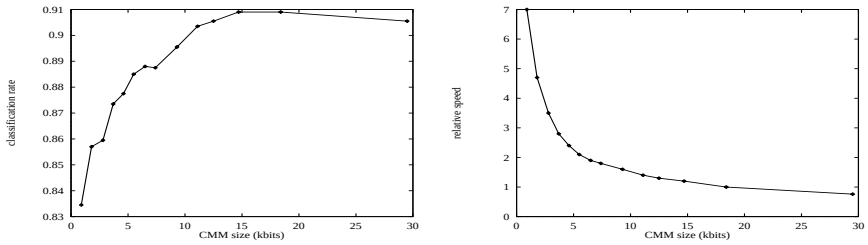


Figure 6: Effects of the CMM size on (a) c-rate and (b) r-speed on the satellite image data

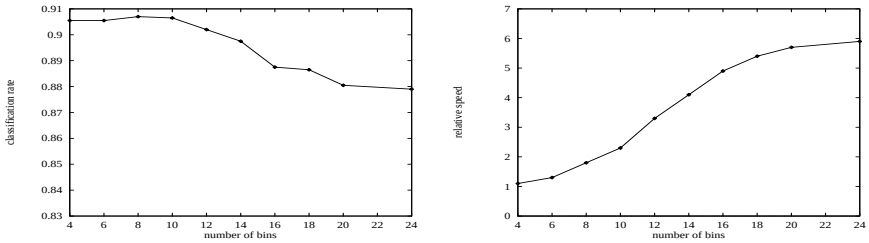


Figure 7: Effects of the number of bins on (a) c-rate and (b) r-speed

Choices of the CMM size and the number of bins N_b may be application dependent, for instance, in favour of the speed or accuracy. In the experiment it was required that the r-speed is not 4 times less and c-rate is not 1% lower than that of the *k*-NN method. Table 2 contains the speeds of the four methods relative to the recall speed of the CMM on the four benchmarks. It is interesting to note that the recall speeds of MLP and RBF networks were 1~25x faster than that of the CMM classifier, but their training speeds were several hundreds times slower. The *k*-NN method needed no training and had the recall speeds 0.043~0.176 times that of the CMM classifier. The overall speed (including training and recall time) of the CMM classifier is over 4 times that of the *k*-NN method. When using the PRESENCE, i.e. the dedicated CMM hardware, the speed of the CMM was further increased over 3 times.

The classification rates by the four methods are given in Table 3, which shows the CMM classifier performed less than 1% less accurate than the *k*-NN method.

	satellite image		image segment		shuttle		Letter	
method	training	test	training	test	training	test	Training	test
MLPN	0.027	5.0	0.004	2.0	0.179	25.25	0.031	14.09
RBFN	0.013	3.57	0.001	1.0	0.078	20.20	0.040	9.69
simple *k*-NN	0	0.176	0	0.111	0	0.043	0	0.146
CMM	2.78	1	2	1	1.85	1	3.60	1

Table 2 Relative training and test speeds of four methods on four benchmarks

	Satellite image	image segment	shuttle	Letter
MLPN	0.914	0.950	0.998	0.923
RBFN	0.914	0.939	0.997	0.941
simple *k*-NN	0.906	0.956	0.999	0.954
CMM	0.901	0.948	0.999	0.945

Table 3 Classification rates of four methods on four benchmarks

The 'two-spirals' benchmark in Figure 8a is interesting as this highly non-linear problem is extremely hard for back-propagation networks and relatively easy for an RBF or Cascade-Correlation net [12]. We found that this task was extremely easy for the CMM. Figure 8c shows that a CMM correctly discriminated all data points, including training and unseen ones.

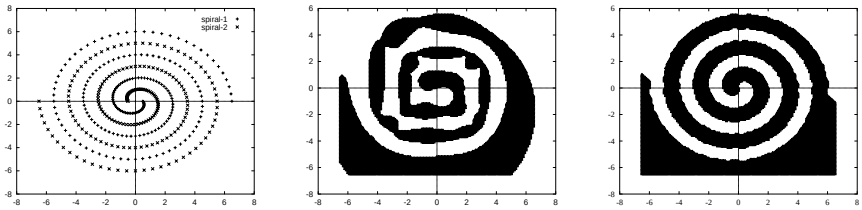


Figure 8: (a) Two spirals, (b) classification by a RBF net and (c) classification by the CMM

5 Conclusions

In this paper we have presented a classifier, which uses a binary CMM for storing and matching a large amount of patterns efficiently, and the *k*-NN rule for classification. The RU encoder converts numerical inputs into binary ones with the maximally achievable uniformity to meet requirements of the CMM. Experimental results on the four benchmarks show that the CMM classifier, compared with the simple *k*-NN method, gave slightly lower classification accuracy, less than 1%, and over 4 times speed-ups in software and 12 times speed-ups in hardware. Therefore our method has resulted in a generic and fast classifier. Compared with MLP and RBF networks, the CMM needs a very short training time. When new training data arrive in an incremental way, MLP and RBF nets needs to be retrained, but with the CMM, the new samples can be simply added to the memory.

This paper has also described a hardware implementation of a FPGA based chip set and a processor card that will support the execution of binary correlation matrix

memories. It has shown the viability of using a simple binary neural network to achieve high processing rates. The approach allows both recognition and training to be achieved at speeds well above two orders of magnitude faster than conventional workstations at a much lower cost than the workstation. The system is scaleable to very large problems with very large weight arrays. Our current research is aimed at showing that the system is scaleable, evaluating methods for the acceleration of the pre- and post processing tasks and considering greater integration of the elements of the processor through VLSI. For more details of the AURA project and the hardware described in this paper see our web page (http://www.cs.york.ac.uk/arch/nn/aura.html).

6 Acknowledgements

We acknowledge the support of British Aerospace and the Engineering and Physical Sciences Research Council (grant no. GR/K 41090 and GR/L 74651) for this research. Our thanks are given to Rick Pack, Anthony Moulds, Zyg Ulanowski, Richard Jennison for the construction and testing of the cards and Ken Lees for discussions and help in preparation of this paper.

References

[1] Willshaw DJ, Buneman OP, Longuet-Higgins HC, 1969, Non-Holographic Associative Memory, *Nature*, Vol. 222, p960-962.

[2] Austin J. AURA, A distributed associative memory for high speed symbolic reasoning. In: Ron Sun (ed), *Connectionist Symbolic Integration.* Kluwer, 1996.

[3] Turner M and Austin J. Matching performance of binary correlation matrix memories. *Neural Networks* 1997; 10:1637-1648.

[4] Turner M and Austin J. A neural network technique for chemical graph matching. 5th *International Conference on Artificial Neural Networks*, Ed. Niranjan M, University of Cambridge, UK, 7-9 July, 1997, Publisher IEE, 187-192.

[5] Fukunaga K. *Introduction to Statistical Pattern Recognition* (charpter 7). 2nd ed., Boston, London: Academic Press, 1990.

[6] Dasarathy BV. Minimal consistent set (MCS) identification for optimal nearest neighbor decision system design. *IEEE Trans. Systems Man Cyb.*, 1994; 24:511-517.

[7] Grother PJ, Candela GT, Blue JL. Fast implementations of nearest neighbor classifiers. *Pattern Recognition*, 1997; 30:459-465.

[8] Austin J, Stonham TJ. An associative memory for use in image recognition and occlusion analysis. *Image and Vision Computing*, 1987; 5:251-261.

[9] Mitchell R, 1997, *A Visually Guided Robot System*, University of York 4th year Project.

[10] Michie D, Spiegelhalter DJ, Taylor CC. *Machine learning, neural and statistical classification* (Chapter 9). New York, Ellis Horwood, 1994.

[11] Zhou P and Austin J. Learning criteria for training neural network classifiers, Accepted by *Neural Computing and Applications Forum*, 1998.

[12] Fahlman SE, Lebiere C. The cascade-correlation learning architecture. In: *Advances in Neural Information Processing Systems*, NIPS 1990; 524-532.