RL-TR-91-262
Final Technical Report
October 1991

(2)

ADVANCED SIGNAL PROCESSING TECHNIQUES

JENS Research

B. Andriamanalimanana, J. Novillo, S. Sengupta

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-91-262 has been reviewed and is approved for publication.

APPROVED:

TERRY W. OXFORD, Captain, USAF
Project Engineer

FOR THE COMMANDER:

GARRY W. BARRINGER
Technical Director
Intelligence & Reconnaissance Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL(IRAP) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October 1991	Final -----

4. TITLE AND SUBTITLE
ADVANCED SIGNAL PROCESSING TECHNIQUES

5. FUNDING NUMBERS
C - F30602-88-D-0026,
Task I-9-4265
PE - 31001G
PR - 3189
TA - 03
WU - P1

6. AUTHOR(S)
B. Andriamanalimanana, J. Novillo, S. Sengupta

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
JENS Research
139 Proctor Blvd
Utica NY 13501

Sub

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory (IRAP)
Griffiss AFB NY 13441-5700

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-91-262

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Terry W. Oxford, Captain, USAF/IRAP/(315) 330-4581
Submitted by JENS Research to Calspan-UB Research Center

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The research efforts under this program have been in the area of parallel and connectionist implementations of signal processing functions such as filtering, transforms, convolution and pattern classification and clustering. The topics which have been found to be the most promising, in as far as their novelty, and applicability to the classification of pulses are covered in the following four sections:

1. Algebraic Transforms and Classifiers
2. An Interaction Model of Neural-Net based Associative Memories
3. Models of Adaptive Neural-Net based Pattern Classifiers
4. Discrete Fourier Transforms on Hypercubes

NOTE: Rome Laboratory/RL (formerly Rome Air Development Center/RADC)

14. SUBJECT TERMS
An Interaction Model of Neural-Net based Associative Memories, Content Addressable Memories, Discrete Fourier Transforms

15. NUMBER OF PAGES
[illegible]

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Advanced Signal Processing Techniques

Technical Summary

Our research efforts have been in the area of parallel and connectionist implementations of signal processing functions such as filtering, transforms, convolution and pattern classification and clustering.

Much more than is reported here was reviewed and entertained for further study and potential implementation. For example, we considered in detail implementation of techniques common to all filtering such as the computation of recursive linear equations using systolic arrays and pipeline and vector processors. Similarly, various neural architectures were studied and implemented. These include Hopfield, Hamming, Back Propagation and Adaptive Resonance Theory Networks.

We report here those topics which we found to be most promising in as far as their novelty and applicability to the classification of pulses and additionally on a new, more transparent method for the computation of the discrete fourier transform on hypercubes. The report consists of four sections:

Algebraic Transforms and Classifiers presents a two-layer neural net classifier with the first layer mapping from feature space to a transform or code space and the second layer performing the decoding to identify the required class label. This

research strongly suggests the further investigation of new types of neural net classifiers whose kernels can be identified with linear and nonlinear algebraic binary codes.

An Interaction Model of Neural-Net based Associative Memories presents a new model for Content Addressable Memories which takes into account pattern interactions generated dynamically over time. It is hoped that this proposed scheme will yield low error rates.

Models of Adaptive Neural-net based Pattern Classifiers deals with design and implementation issues at the operational level of ART-type parallel distributed architectures in which learning is asynchronously mediated via a set of concurrent group nodes comprising a number of elemental processors working on component substrings of the input pattern strings. A single-slab Bland-net alternative architecture is also proposed.

Discrete Fourier Transforms on Hypercubes presents a decomposition of the transform matrix that is easier to understand than others found in the literature and a basic scheme for power-of-prime length transform computations on the hypercube. For lengths that are not a power of a prime, the use of the Chinese Remainder Theorem in pipelining the computation is proposed and discussed.

ALGEBRAIC TRANSFORMS AND CLASSIFIERS

A two-layer neural net classifier is proposed and studied. Codewords from a {1,-1} algebraic code are "wired" in the net to correspond to the class labels of the classification problem studied. The first layer maps from feature space to some hypercube $\{1,-1\}^n$, while the second layer performs algebraic decoding in $\{1,-1\}^n$ to identify the codeword corresponding to the required class label.

The main motivation for introducing the proposed architecture is the existence of a large body of knowledge on algebraic codes that has not been used much in classification problems. It is expected that classifiers with low error rates can be built through the use of algebraic coding and decoding.

1. Neural network classifiers

Artificial neural nets have become the subject of intense research for their potential contribution to the area of pattern classification [3]. The goals of neural net classifier construction are manifold and include the building of lower error rate classifiers than classical (i.e Bayesian) classifiers, which are capable of good generalization from training data, and having only reasonable memory and training time requirements. Most neural net classifiers proposed in the literature adjust their parameters using some training data. In this proposal, training has been set aside to bring out possible benefits of algebraic coding. In subsequent work, classifier parameter adjustment will be studied in conjunction with the algebraic coding approach.

2. Net topology and operation

The neural net consists of three layers of units connected in a feedforward manner.

The number of units in the input layer is the length k of each feature vector. The hidden layer has as many units as the length of the algebraic code used. Finally the output layer

contain one unit for each class in the classification problem. The chosen codewords from the algebraic code used are wired into the net via the interconnection weights between the input and the hidden layers. The decoding properties of the algebraic code come into play in the interconnection weights between the hidden and the ouput layers. The actual choice for these weights will be detailed below. Note that even though the topology chosen here resembles other popular neural net topologies, such as that of the back-propagation network, this first version of our classifier is non-adaptive and non-learning (the pattern classes are not learned but wired in from some a priori knowledge), and the time needed to recognize a pattern is only the propagation delay through the two layers of weights.

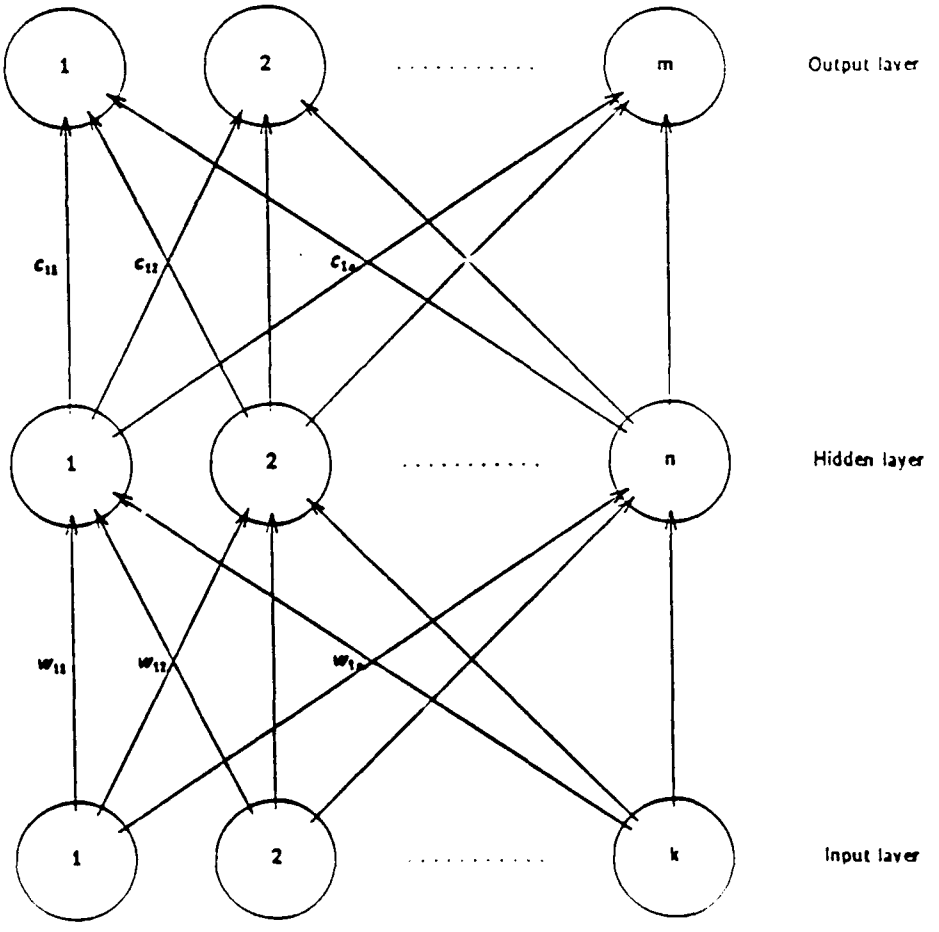


Figure 1

The network operates as follows. A feature vector f to be classified is presented to the input layer; this feature vector is a corrupted version of some class representative l (the latter associated with the codeword c). The second layer of units outputs a vector u, the 'frequency domain' transform of the feature vector f, which is a corrupted version of c. The output layer decodes u into the codeword c, thereby identifying the class represented by l. The output layer does this by having all units 'off' except the one corresponding to codeword c, i.e class l.

To make the net operate as required, we need to devise schemes for wiring codewords in the interconnection weights so as to achieve a 'continuous' mapping from feature space to code space (feature vectors that are close correspond to vectors in $\{1,-1\}^n$ that are also close; this is important if the error rate is to be kept low), decide the transformation performed by each unit and find a way of making the decoding of a corrupted codeword correspond to turning on exactly one unit in the output layer. Finally, we must analyze the kind of algebraic codes that are suitable for the proposed architecture.

First, note that the input layer of units will simply buffer the feature vector to be classified; hence the units here perform the identity transformation. Secondly, it should be clear that the units in the hidden and output layers must be polar (i.e they output 1 or -1; actually we could have made them binary but it is easier to deal with polar outputs for neural nets). We have decided to make them threshold units, outputting +1 if the net input is at least the threshold θ of the unit, -1 otherwise. The value of θ will be determined below. But first we need to make some elementary observations about $\{1,-1\}$ codes.

3. Remarks on {1,-1} codes

We remark that if c and c' are two $\{1,-1\}$ vectors of the same length n, then

$$c \cdot c' + 2\,\text{distance}(c,c') = n$$

where the dot is the usual dot product and distance is the Hamming distance between the

two vectors. This is so because for {1,-1} vectors, c . c' is the number of coordinates where the two vectors agree minus the number of coordinates where they disagree; hence c . c' is n minus twice the Hamming distance of the two vectors.

A simple consequence of this observation is that if C is a {1,-1} code of length n, i.e a collection of distinct {1,-1} vectors all of length n, and if M is the maximum dot product among different codewords of C, then the minimum Hamming distance among different vectors of C is

$$0.5 * (n - M)$$

and thus the code corrects any vector corrupted by less than

$$0.25 * (n - M)$$

errors, using the usual nearest neighbor decoding scheme. Because of this, we will be looking for {1,-1} codes where the length n is much larger than the maximum dot product M.

Next, we remark that if c is the closest codeword in the code C to a {1,-1} vector u, and if w is any other vector in C, then the following inequalities hold

$$w . u < 0.5 * (n + M) < c . u$$

This can be seen as follows. Let $u \in \{1,-1\}^n$ and let c be the codeword in C closest to u (note that at first sight there may be several codewords in C having minimum distance from u; however if u has not been corrupted by 0.25 * (n - M) or more errors, then this is impossible!; we will assume this from now on).

Now let u = c + e , where the error vector e has its q nonzero components equal to 2 or -2. We have

$$\text{distance}(u,w) > \text{distance}(u,c), \text{ for all } w \in C - \{c\}$$

Now

$$c . u = n - 2\,\text{distance}(c,u) = n - 2q$$

Since it is assumed that less than 0.25 * (n - M) errors have been made in corrupting c to get u, we have

$$q < 0.25 * (n - M)$$

Hence

$$c \cdot u = n - 2q > n - 0.5 * (n - M) = 0.5 * (n + M)$$

Also,

$$\begin{aligned} w \cdot u &= w \cdot (c + e) \\ &= w \cdot c + w \cdot e \\ &\leq M + 2q \text{ , because the q nonzero components of e are 2 and -2} \\ &< M + 0.5 * (n - M) = 0.5 * (n + M) \end{aligned}$$

Hence the claim is proved.

Note that the last observation will be used to decide the threshold for the units in the output layer, and the interconnection weights between the hidden and output layers.

4. Weight matrix from hidden to output layers

Choose a suitable code C (suitable in the sense of having the length n much larger than the maximum dot product M) and select m codewords

$$c_1, c_2, \cdots, c_m$$

from the C. Writing the components of c_i as

$$c_{i1}, c_{i2}, \cdots, c_{in}$$

use the following matrix as the weight matrix from the hidden to output layers

$$C_{m \times n} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ . & . & \cdots & . \\ . & . & \cdots & . \\ . & . & \cdots & . \\ c_{m1} & c_{m2} & \cdots & c_{mn} \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ . \\ . \\ . \\ c_m \end{bmatrix}$$

In other words, the weights from the hidden layer to the output layer are all 1 or -1, and the weights from the n hidden units to the jth output unit are the components of the codeword c_j (see **Figure 1**).

Now if the output of the hidden layer is the vector u, then the net input to the jth output unit is

$$c_j \cdot u$$

Hence if the threshold of each unit in the output layer is chosen to be

$$0.5 * (n + M)$$

then by the observation in section **3.** only the unit corresponding to the codeword closest to u will output 1, the rest -1.

5. Weight matrix from input to hidden layers

The threshold in the units of the hidden layer are chosen to be 0. Now let W_{kxn} be the weight matrix between the input and the hidden layers. Let R_{mxk} be the matrix whose rows are the class labels in the classification problem at hand.

$$R_{m\,x\,k} = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1k} \\ r_{21} & r_{22} & \cdots & r_{2k} \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ r_{m1} & r_{m2} & \cdots & r_{mk} \end{bmatrix} = \begin{bmatrix} r^1 \\ r^2 \\ \cdot \\ \cdot \\ \cdot \\ r^m \end{bmatrix}$$

Several schemes can be imagined for computing the matrix W_{kxn} from the matrices R_{mxk} and C_{mxn}.

The first scheme is a simple correlation, defined by

$$W_{kxn} = R_{mxk}{}^{T} C_{mxn}$$

where $R_{mxk}{}^{T}$ denotes the transpose of the matrix R_{mxk}. In other words, the weight from the ith input unit to the jth hidden unit is the sum of the correlations between the ith component of a class label and the jth component of the corresponding codeword (see also **Figure 1**).

$$w_{ij} = \sum_{l=1}^{m} r_{li} c_{lj}$$

The idea here is a natural one. In order to achieve continuity, we seek to strengthen the connection between a class label and its corresponding codeword.

Another scheme 'simulates' a linear transformation from feature space to code space. If the hypercube $\{1,-1\}^n$ were a vector space, which it is not, and if the class labels, i.e the rows of the matrix R_{mxk}, were linearly independent, then there would be a unique linear transformation mapping class label r^i onto codeword c_i, for all i with $1 \leq i \leq m$. In other words, there would be a unique kxn matrix W with

$$R_{mxk} \cdot W = C_{mxn}$$

If R were square, it would be nonsingular and we could write

$$W = (R_{mxk})^{-1} \cdot C_{mxn}$$

which would give the interconnection weights we are looking for (recall a linear transformation between finite vector spaces is always continuous).

Of course, the above assumptions do not hold in most cases. We may though use some kind of approximation of the inverse for R, called the pseudoinverse of R and denoted R^* (we have dropped the index mxk for simplicity).

If the matrix R is of maximum rank, but not square, then $R \cdot R^T$ is nonsingular and the pseudoinverse $\bar{R}$ is defined by

$$\bar{R} = R^T \cdot (R \cdot R^T)^{-1}$$

If $R \cdot R^T$ is not invertible, then $\bar{R}$ is given by a limit

$$\bar{R} = \lim (R^T \cdot (R \cdot R^T + \epsilon^2 I)^{-1}) \text{ as } \epsilon \rightarrow 0$$

where I denotes the identity matrix of the size of $R \cdot R^T$ and ϵ is a positive real number.

It can be shown that the pseudoinverse $\bar{R}$ always exists. Of course, in practice, we only compute an approximation of the limit most of the time, since the exact form of $\bar{R}$ may be difficult to obtain. For a nonsingular matrix, the ordinary inverse and the pseudoinverse are of course identical.

In any case, we can take as weight matrix from the input layer to the hidden layer the product

$$W_{kxn} = \bar{R} \cdot C_{mxn}$$

6. Hadamard coding

It is believed that the error rate of the proposed classifier depends heavily on the number of errors that the algebraic code wired into the net can correct, which, as we have seen, is a linear function of (n - M), where n is the length of the code and M is the maximum dot product. We then need to investigate {1,-1} codes for which n is significantly larger than M.

The class of Hadamard codes seems to provide a reasonable starting point. Recall that Hadamard matrices constitute a handy tool in many disciplines, including signal processing (Hadamard matrices are the kernel of the so-called Hadamard-Walsh transform) and design of experiments (Hadamard matrices give rise to good symmetric combinatorial designs). Even though Hadamard matrices exist only for certain orders, we believe that they are still quite useful for the classifier architecture proposed.

A Hadamard matrix of order n is a square matrix H of order n, whose entries are 1 or -1, and such that

$$H \cdot H^T = n\, I$$

where I is the identity matrix of order n.

The definition really says that the dot product of any two distinct rows of H is 0, hence if we take as codewords all the rows of H, then M = 0, i.e the code can tolerate any number of errors less than 0.25n. Of course the threshold of each output unit of the classifier must be set to 0.5n.

Examples of low-order Hadamard matrices are shown in **Figure 2**, where -1 is simply written -.

$$n = 1:\ H_1 = \begin{bmatrix} 1 \end{bmatrix} \qquad n = 2:\ H_2 = \begin{bmatrix} 1 & 1 \\ 1 & - \end{bmatrix}$$

$$n = 4:\ H_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & - & 1 & - \\ 1 & 1 & - & - \\ 1 & - & - & 1 \end{bmatrix}$$

$$n = 8:\ H_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & - & 1 & - & 1 & - & 1 & - \\ 1 & 1 & - & - & 1 & 1 & - & - \\ 1 & - & - & 1 & 1 & - & - & 1 \\ 1 & 1 & 1 & 1 & - & - & - & - \\ 1 & - & 1 & - & - & 1 & - & 1 \\ 1 & 1 & - & - & - & - & 1 & 1 \\ 1 & - & - & 1 & - & 1 & 1 & - \end{bmatrix}$$

Figure 2

It can easily be shown that a Hadamard matrix of order n can exist only if n = 1, 2 or n = 0 mod 4. It has not yet been proved that a Hadamard matrix of order n exists whenever n is a mutiple of 4, even though this is widely believed.

For our purposes it is enough to note that Hadamard matrices do exist for orders that are a power of 2. This is seen through the fact that if H_n is a Hadamard matrix of order n, then H_{2n} is a Hadamard matrix of order 2n, where

$$H_{2n} = \begin{bmatrix} H_n & H_n \\ H_n & -H_n \end{bmatrix}$$

7. Simplex coding

Recall the construction of the finite field $GF(p^k)$ with p^k elements and prime subfield $GF(p) = Z_p$, where p is a prime number (p = 2 is the most important case for our classifier). An irreducible polynomial h(x) with coefficients in GF(p) and of degree k is chosen. The elements of $GF(p^k)$ are taken to be all polynomials with coefficients in GF(p) and of degree less than k. Addition and multiplication are ordinary addition and multiplication of polynomials but performed modulo h(x). It can easily be proved that indeed $GF(p^k)$ satisfies all axioms of a field and that it contains p^k elements.

It is well-known that the multiplicative group $(GF(p^k) - \{0\}, *)$ of $GF(p^k)$ is cyclic of order $p^k - 1$. If x, as element of the field, is a generator of this cyclic group, then h(x) is called a primitive polynomial over GF(p). Note that databases of primitive polynomials over GF(p) are available for many values of p.

Let

$$h(x) = x^k + h_{k-1}x^{k-1} + h_{k-2}x^{k-2} + \ldots + h_1 + h_0$$

be a primitive polynomial of degree k over GF(2). So $h_i = 0$ or 1 for all i and the irreducibility of h(x) implies that $h_0 = 1$.

We define the pseudo-noise sequence generated by h(x) and given bits $a_0, a_1, \ldots, a_{k-1}$ as the binary sequence

$$a_0, a_1, \ldots, a_{k-1}, a_k, a_{k+1}, \cdots$$

where, for $l \geq k$,

$$a_l = a_{l-k} + a_{l-k+1}h_1 + a_{l-k+2}h_2 + a_{l-1}h_{k-1}$$

(addition and multiplication are over GF(2), i.e modulo 2).

The following properties are well-known [4]:

(i) Let $C = a_l a_{l+1} a_{l+2} \cdots a_{l+n-1}$ be a subsequence of length n of a pseudo-noise sequence (a_i) generated by h(x), where

$$n = 2^k - 1$$

Then for any cyclic shift C' of C, C + C' is another cyclic shift of C.

(ii) If C and n are as in (i) then C has 2^{k-1} ones and $2^{k-1} - 1$ zeros.

Now from a psudo-noise sequence generated by h(x), a binary code of length $n = 2^k - 1$, denoted C^-, is constructed by taking as codewords the vector

$$a_0, a_1, \ldots, a_{2^k-2}$$

and all of its $2^k - 2$ cyclic shifts.

The code C^- together with the all-zero vector of length $2^k - 1$ is also known in the literature as the cyclic simplex code of length $2^k - 1$ and dimension k. By property (i) above, the simplex code is a vector subspace of dimension k of $GF(2)^n$, where $n = 2^k - 1$.

From C^- a {1,-1} code C is constructed, through the affine mapping $a \rightarrow 1 - 2a$ (i.e replace 0 by 1 and 1 by -1).

For u, w $\in$ C, we can compute u.w as follows.

If u = w then clearly $u.w = n = 2^k - 1$.

If $u \neq w$ then u.w = n - 2 distance(u,w) as explained in section 3. But distance(u,w) is the distance between the corresponding vectors in C^-, which by linearity of the simplex code $C^- \cup \{0\}$, equals

$$\text{distance}\,(0, \bar{c}_2 - \bar{c}_1)$$
$$= \text{distance}\,(0, \bar{c}_2 + \bar{c}_1)$$
$$= \text{weight}\,(\bar{c}_2 + \bar{c}_1)$$
$$= 2^{k-1} \text{ by property (ii) above.}$$

So,

$$u.w = n - 2\,\text{distance}(u,w)$$
$$= 2^k - 1 - 2 \cdot 2^{k-1} = -1$$

Hence for the code C, the maximum dot product is $M = -1$. C can tolerate any number of errors less than $0.25 * (n+1) = 2^{k-2}$. The threshold of each output unit of the classifier must be set to $0.5 * (n+1) = 2^{k-1}$.

We give a couple of examples to illustrate the construction.

For $h(x) = x^3 + x + 1$, and $a_0 = 0$, $a_1 = 0$ and $a_2 = 1$, we get

$$a_3 = a_0 + a_1 \cdot h_1 + a_2 \cdot h_2 = 0$$
$$a_4 = a_1 + a_2 \cdot h_1 + a_3 \cdot h_2 = 1$$
$$a_5 = a_2 + a_3 \cdot h_1 + a_4 \cdot h_2 = 1$$
$$a_6 = a_3 + a_4 \cdot h_1 + a_5 \cdot h_2 = 1$$

The codes $\bar{C}$ and C are

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & - & 1 & - & - & - \\ - & 1 & 1 & - & 1 & - & - \\ - & - & 1 & 1 & - & 1 & - \\ - & - & - & 1 & 1 & - & 1 \\ 1 & - & - & - & 1 & 1 & - \\ - & 1 & - & - & - & 1 & 1 \\ 1 & - & 1 & - & - & - & 1 \end{bmatrix}$$

for which we can see that the minimum distance is indeed 4.

Another example is provided by $h(x) = x^4 + x + 1$, $a_0 = a_1 = a_2 = 0$ and $a_3 = 1$. We compute

$$a_4a_5a_6a_7a_8a_9a_{10}a_{11}a_{12}a_{13}a_{14} = 00110101111$$

The code $\tilde{C}$ is shown below, for which the {1,-1} code C is easily obtained.

000100110101111
100010011010111
110001001101011
111000100110101
111100010011010
011110001001101
101111000100110
010111100010011
101011110001001
110101111000100
011010111100010
001101011110001
100110101111000
010011010111100
001001101011110

8. Future work

We need to analyze more classes of algebraic codes suitable for the proposed architecture. One of the promising directions is the area of codes from block designs (as mentioned earlier, the Hadamard codes actually fall under this area). Properties of these codes are often distilled from the geometric properties of the designs. Work in this area is in progress.

Next we should actually attempt an analytical study of the error rate of the proposed

classifier. Note that the mapping from feature space to code space may actually introduce more errors. Even if this mapping does not introduce any error, it may still be difficult to derive a closed-form of the error rate, so bounds may be the best we can do.

Finally we need to design an adaptive version of the proposed architecture, where the code is somehow wired in the net but a bettering of the estimates of the parameters of the classifier is also accomplished through training data.

9. References

[1] Hopfield,J.J.,Neural networks and physical systems with emergent collective computational abilities, Proc. Nat'l. Acad. Sci., USA, v.79, pp. 2554-2558, 1982.

[2] Lippmann,Richard P., An introduction to computing with neural nets, IEEE Acoustics Speech Signal Processing Magazine, v. 4, pp. 4-22, 1987.

[3] Lippmann,Richard P., Pattern classification using neural networks, IEEE Communications Magazine, November 1989, vol. 27, no. 11.

[4] MacWilliams, F.J and N.J.A. Sloane, The theory of error-correcting codes, 2 volumes, North-Holland, 1977.

[5] Widrow, B. and R. Winter, Neural nets for adaptive filtering and adaptive pattern recognition, Computer, v. 21, no. 3, 1988.

An Interaction Model of Neural-Net based Associated Memories

1. Introduction

One of the most focused topic in the entire neural network paradigm that still continues engagement of major research effort is evidently the topic of Associative or Content-Addressable Memory (CAM) in the context of a Connectionist framework (9). An Associative Memory (AM) or a Content-Addressable Memory is a memory unit that maps an input pattern or a feature vector $x_k \in \mathbf{R}_n$ to an associated output vector $y_k \in \mathbf{R}_m$ in O(1) time. The input pattern x_k may be a substring of the desired output pattern y_k, or may be totally distinct; however, in both cases, we identify an association of the form (x_k, y_k) for each pattern index k. An AM or CAM is a storage of all such association pairs over a given pattern domain. The input string is also known as the cue-vector, or a stimulus, the output string, the associated response pattern.

Viewing it from a neural network perspective, the ensemble, even if conceptualized as a pattern classifier/recognizer in a strict functional sense, is essentially an AM/CAM entity if it works correctly. Because ultimately, a neural-net based classifier/recognizer logically maps an external object via an appropriate feature space onto distinct classes, we could say that in a functional context (object → input pattern (stimulus), pattern-class) comprise associative pairs. In this section we present a new model of Associated Memories or Content-addressable Memories realizable, say, via an appropriate optical implementation of an artificial neural net type system using lens, masks, gratings, etc (4,6). The issue addressed here is not this realization task *per se*, but the more important issue of models basically congruent to neural net type systems from the point of view of functionality and performance, the models which reflect distributivity, robustness, fault-tolerance, etc. The implementation issue has to be addressed eventually – one could aim for a neural net type system in an electronic (as opposed to optical) environment but that as a topic must wait till one clears out the model related issues, the overall architectural framework of achieving a certain set of objectives at the topmost level of specification.

In this paper, a matrix model of Associative memory is proposed. The strength of the model lies in the way it is different from the traditional AM models articulated thus far (see Hopfield (7,11), Kohonen (8,9) etc. for Associative Memory models). This model becomes significant at high storage density of patterns when the logically neighboring patterns do tend to perturb each other considerably. A memory system even locally stable in the sense of Lyapunov at or around specific logical sites may tend to display erraticity upon receiving another pattern for storage if the pattern storage density increases (1,2,3). In such cases, instead of trying to keep patterns away from interacting with each other, one could deliberately allow them to interact and then consider all such collective interactions in the overall dynamics of the system. To date, no such model has been proposed.

Our model is different from the proposed classical memory models in one significant sense. It allows interactions among patterns to perturb the associated feature space. As we pack dynamically more and more patterns into the system, more skewed becomes the "neighborhood" of a pattern. It is analogous to the situation where an electric field around

a charged element is peturbed by bringing in more charge particles in the vicinity. It is this recognition in our model that provides a changed perspective.

We observe at the outset the two salient features associated with neural net type processors or systems using such processors. First, a neural network model is ultimately a content-addressable memory (CAM) or even an an associative memory (AM) depending upon how it is implemented. The central theme in such models is storing of information of the relevant patterns distributively in a holistic fashion over the synaptic weights between pairs of neuron-like processor elements allowing both storage and retrieval tasks more or less in a fail-soft manner. Even though the time-constant of a biological neuron's switching time is of the order of milliseconds, storage and recall of complex patterns are usually carried out speedily in a parallel distributed processing mode. It is ultimately this notion of distributed memory in contrast to local memory of conventional computers that stands out in so far as neural net type systems are concerned.

A neural network system could also be viewed at the same time as a pattern classifier/recognizer. Problems like Radar target classification, object identification from received sonar signals are where neural net type systems are most adept. In particular, if a recognition of a partially viewed object or a partially described object has to be made, if the data is further contaminated with white noise then not only a neural-based classifier appears to be more promising for real-time classification from the point of view of performance but a CAM or an AM oriented system might just be ideal. It depends, of course, on how we ultimately structure the problem domain, what type of cues are allowed for appropriate responses, etc. but it is eminently possible to lean on a specific CAM/AM oriented neural-net paradigm to do just that. In this section, a *generalized outer product* model is proposed.

The associated-memory elements ought to yield a performance close to O(1) search time. However, once pattern-pattern interactions are allowed within the model one cannot guarantee O(1) performance. In our model, it is feasible that on some cue patterns the problem is so straight forward that one need not consider pattern-pattern interaction at all. In such cases, it behaves as classical AM element with O(1) search-time.

Consider an orthonormal basis $\{\phi_k\}$ spanning an L-dimensional Euclidean vector space in which associated pairs of patterns (cue, response) in the form (x_k, y_k) are stored in correlation-matrix format of association. The whole idea is to buffer the stored patterns from the noisy cue vecors via this basis such that one could see the logical association between x and y via a third entity, our basis. Note that a cue vector x_k in this basis is expressed as

$$|x_k> = \sum_i \alpha_{ki} |\phi_i>$$

with its transpose as

$$<x_k| = \sum_i \alpha_{ki} <\phi_i|$$

such that the dot product between two such vectors $|x_k>$ and $|x_l>$ is

$$<x_k|x_l> = \sum_{ij} \alpha_{ki}\,\alpha_{lj}\,\delta_{ij}$$

$$= \sum_{i} \alpha_{ki}\ \alpha_{li} = a_{kl}$$

The Euclidean norm of a vector is then

$$<x_k|\,x_k> = \sum_{i} (\alpha_{ki})^2 = a_{kk}$$

Note that the cue-vectors themselves, in this model, need not constitute an orthogonal vector space. Even though this is a desirable assumption to consider, in reality, it may not be a good assumption to depend on. Secondly, the vector space $\{\phi\}$ need not be *complete*, an appropriate subsapce would do as indicated by Oja and Kohonen (13). They build the pattern space over some p-dimensional subspace with $p \leq L$ and deal with a pattern by having its projection on this subspace and extrapolating its residual as an orthogonal projection on this space. We could also extend our model in this way.

We could interpret the coefficients α_{ki} associated with a cue-feature as follows. The *normalized coefficient* $\alpha_{ki}/\sum_{j} \alpha_{kj}^{\,2}$ could be regarded in this model as the probability amplitude that the vector $|x_k>$ has a nonzero projection on the elemental basis vector $|\phi_i>$. The square of such terms are the associated probabilities; if we want to picture it a cue vector in this representation would be a sequence of spikes on the ϕ-domain whose height at any specific ϕ-point is the probability amplitude that the vector there has a nonzero projection. Note that the sum of the square of all these projections must necessarily equal to 1.0 for a nonzero cue-vector.

Given (cue,response) pairs we form matrix associative memory operators as follows:

$$\mathbf{M}_s = \sum_{k} |\phi_k><x_k|$$

$$\mathbf{M}_r = \sum_{k} |y_k><\phi_k|$$

The operator $\mathbf{M}_s$ on $|x_k>$ cue is

$$\mathbf{M}_s\,|x_l> = \sum_{k} |\phi_k><x_k|x_l>$$

$$= |\phi_l>\,a_{ll} + \sum_{k\neq l} a_{kl}\,|\phi_k$$

Such that its projection on any arbitrary $|\phi_q>$ is

$$<\phi_q | M_s | x_l> = a_{ll}\ \delta_{ql} + a_{ql}$$

We call this the figure of merit of a cue x_l on a basis vector ϕ_q, $fm(x_l | \phi_q)$.

Obviously, the memory operator has the largest projection when q=l, i.e.

$$\text{index } \{\max_q \{ fm(x_l | \phi_q) \} = l$$

Note that if $\max_s \{ fm(x_l|\phi_s), s \neq q \} \simeq a_{ll}$, then the cue x_l may not be discernible from another cue associated with which is an altogether different response. We denote the associated domain of certainty by a resolution measure $res(y_s)$, where

$$res(y_s) = res(y_s | \phi_s)$$

$$= 1 - \frac{1}{a_{ll}} \{\max_{s \neq q} [fm\ (x_l|\phi_s)]$$

In the simplest case, one assumes res(.) on associated pattern vectors to be very large. Or, equivalently, one assumes that the storage of multiple (cue,response) pairs is facilitated by the requirement that the Hamming distance between any two pair of such cues in the library is relatively large. In such cases, after receiving the index value l associated with the given cue $|x_l>$, one continues by operating M_r on $|\phi_l>$. Then

$$M_r |\phi_l> = \sum_k |y_k><\phi_k|\phi_l> = |y_l>$$

indicating successful retrieval of the associated pattern $|y_l>$.

Note that, in general, a received stimulus may not appear in the form as one desires. It could be a noise contaminated incomplete cue-function of the form

$$|x> = \gamma |x_l> + |\delta>, \text{ where } <x_l | \delta> = 0$$

and, $0 < \gamma < 1$. Then a memory operation on x is

$$M_s |x> = \gamma\, M_s |x_l> + M_s |\delta>$$

$$= \gamma a_{ll} |\phi_l> + \gamma \sum_{k \neq l} a_{kl} |\phi_k> + \sum_k |\phi_k><x_k|\delta>$$

Now, given $\delta = \sum_{i \neq l} \xi_i |\phi_i>$ we obtain

$$<x_k | \delta> = \sum_{i \neq l} \xi_i \alpha_{ki}$$

$$<\phi_m | M_s | x> = \gamma (a_{mm} + a_{ml}) + \sum_{i \neq l} \xi_i \alpha_{ki}$$

The matrix elements show deterioration of performance. Since, the cue-vectors need not, in general, form an orthonormal basis the first term on the right hand side of the matrix element would, in general, contain the usual cross-talk component, but now, in addition to this, the distortion δ, even though orthogonal to the most likely cue-vector x_l, would introduce additional degradation.

The question of *optimization* of the memory space is now the issue. Given the possibility of receiving distorted signals at input ports whence one obtains the cue-vector in question, one may approach the design problem in a number of ways.

One simple approach, particularly when storage density is sparse, is to consider the possibility that the received cue x is close to one of the cues considered acceptable. Under this assumption, we form both the on and the off-diagonal terms $<\phi_q| M_s| x>$ and note the index q for which the matrix element is largest. We then logically associate x with y_q on the assumption that y_q is what is stored logically at the basis ϕ_q. The task of retrieving the associated pattern given a cue is then a straightforward problem as long as a relatively sparsely dense set of (cue,response) pairs is associatively stored. The problem occurs when cues are close to each other or when one finds that a single level discriminant may not suffice. If an unknown cue x is not stored precisely in the format it has been received, if it is equally close to x_q as it is to x_r, with $q \neq r$, we must have further information to break the impasse.

One simple strategy is to extend the proposed abstract AM model as follows. In this case, we do not store just single instance (cue, response) items but, instead, a class of items. In other words, we prepare a class-associative storage (or a class-content addressable storage) in the form below

$$C_k = \{(x_{1k}, y_k), (x_{2k}, y_k), ..., (x_{jk}, y_k), ... \}$$

$$M_{cs} = \sum_{ik} |\phi_k><x_{ik}|, \text{ and}$$

$$M_{cr} = \sum_{k} |y_k><\phi_k|$$

Though the essential structure of the tuple (cue_1, cue_2, ..., response) is simple enough it could be exploited in a number of ways allowing us to extend the scope of simple CAM/AM based storage/retrieval task either through a spatial or a temporal extension, or a combination of both.

In the sequel, we conjecture some simple schemes.

A. Spatial Extension.

1. Concurrently, obtain n input signals $\{x_i, 1 \leq i \leq n\}$. These are n different instances of same measure or different measures of same feature element. The

underlying hypothesis is that all these x_i measures point to essentially the same response vector y.

2. Obtain, in parallel, (x_i, y_i).

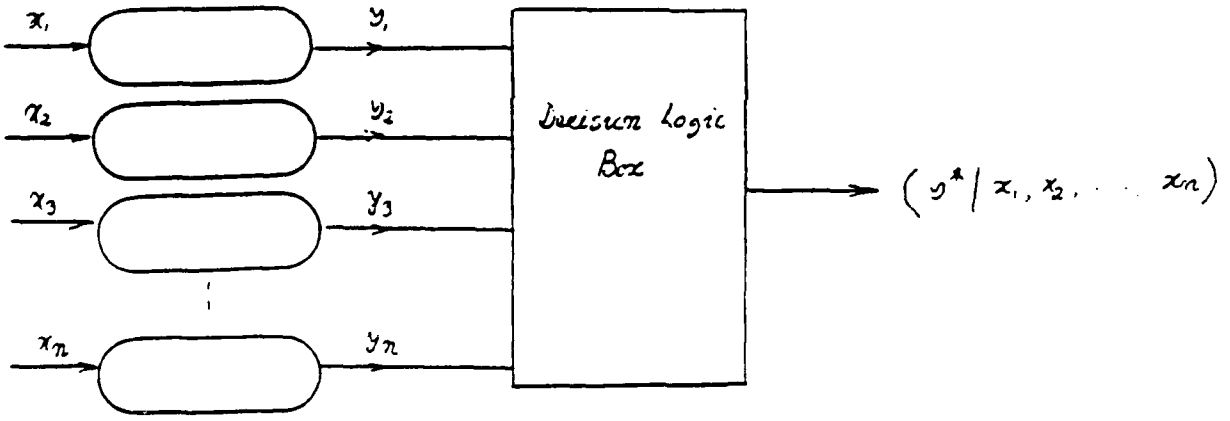
3. At next cycle, at the next higher level, obtain $\eta(y_i)$ for every occurrence of y_i. $\eta(.)$ yields the number of occurrence of y_i.

4. Obtain $y^* = \max_{y_i} \{\eta(y_i)\varsigma(x_i)\}$ where

$$\varsigma(x_i) = \max_{\phi_q} \mathrm{fm}(x_i|\,\phi_q)$$

5. Output $(y^* \,|\, x_1, x_2, \ldots, x_n)$

In this case a concurrent evaluation is proposed after the input stage. At the next layer, a decision logic gate at the output port yields the optimum desired response.



Obviously, this simple design is robust, distributed and modular. It also provides multiple modular redundancy. In the next proposed extension, we introduce a model which is essentially extensive in the temporal domain.

B. Temporal Extension

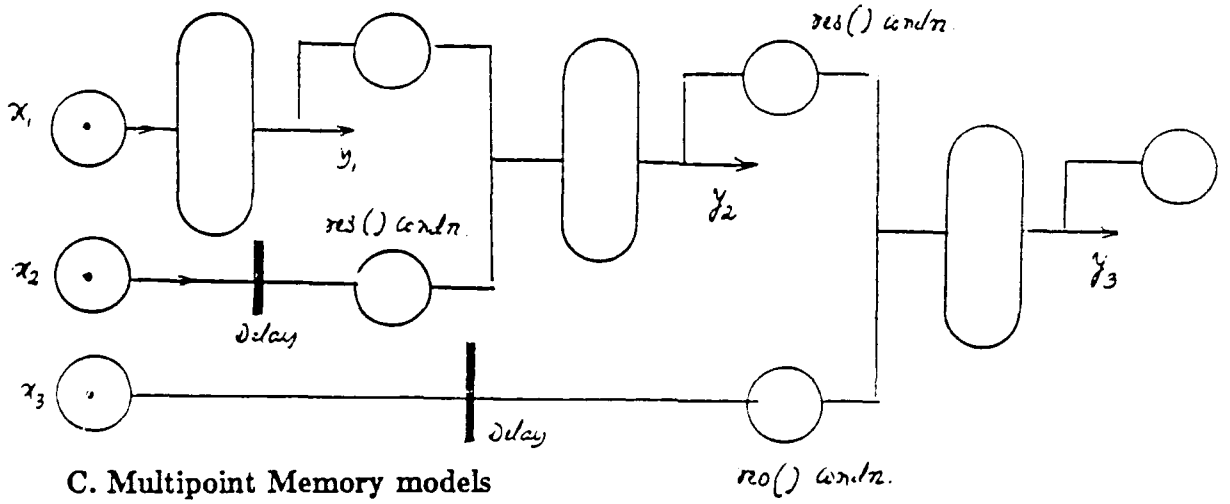
1. $i \leftarrow 1$
2. At cycle i, obtain (x_i, y_i)
3. While $\mathrm{res}(y_i) < \delta$ do

   3.1 $i \leftarrow i+1$
   3.2 obtain at the ith cycle
   $((x_i, y_i) \,|\, \mathrm{res}(y_{i-1}) < \delta, \mathrm{res}(y_{i-2}) < \delta, \ldots, \mathrm{res}(y_1) < \delta)$

3.3 compute res(y_i)

4. Output (y_i^* | (x_{i-1}, y_{i-1}), ... ,(x_1, y_1))

Here the crucial element is the retrieval process specified in 3.2. At the ith cycle, the scheduled retrieval process is in WAIT state until it is fired by the appropriate event at the (i-1)th cycle, namely the arrival of the condition res(y_{i-1}) < δ. The two schemes are outlined below.



C. Multipoint Memory models

Next level of extension is conjectured on a different basis. In this case, we form memories based on pair-wise product form of original basis vectors $|\phi_i>$, $|\phi_j>$ as shown below. We call this a two-point memory system, a special case of the more general multipoint memory system which we propose shortly.

$$M_p^2 = \sum_{kl} \alpha_{kl}\, \psi_{kl} - \beta_{kl}\, \psi_{lk}$$

w ere,

$$\psi_{kl} = |\phi_k(1)\ \phi_l(2)><x_k(1)\ x_l(2)|$$

$$\psi_{lk} = |\phi_l(1)\ \phi_k(2)><x_k(1)\ x_l(2)|$$

In this scheme, one argues that

- Logically, the cue-vector | x_k > is associated with the basis vector $|\phi_k>$, and $|x_l>$ with $|\phi_l>$ with a joint probability α_{kl}.

- However, closer the cue vectors $|x_k>$ and $|x_l>$ are more is the error one is likely to make in this association. In view of probable misclassification, the weight on the initial proposition must therefore be reduced. This we do by considering the counter proposal that one could have $|x_k>$ associated with $|\phi_l>$ and $|x_l>$ with $|\phi_k>$.

respectively, with a probability β_{kl}, instead. Higher is our conviction in this regard (higher b_{kl} values) lower becomes the strength of our original hypothesis.

In this model, the matrix elements assume the forms

$$<\phi_m(1)\ \phi_n(2)\,|\,M_p^2\,|\,x_p(1)\ x_q(2)> = \alpha_{mn}\ a_{mp}\ a_{nq} - \beta_{mn}\ a_{np}\ a_{mq}$$

Given an arbitrary cue vector x, let us assume that res $(y \mid x) < \delta$, and that within a threshold limit the cue-vector x is similar to other cue-vectors whose logical associations are, however, different. That is, we suppose an equivalence class associated with x

$$C = \{\ (x_1,y_1 \mid x \approx x_1),\ (x_2,y_2 \mid x \approx x_2),\ \ldots\ ,\ (x_n,y_n \mid x \approx x_n)\ \}$$

Given this, we would like to know how best one infers the corresponding associated vector y with x. Let us denote the matrix element

$$<\phi_m(1)\ \phi_n(2)\,|\,M_p^2\,|\,x(1)\ x_q(2)> \ =\ (m,n,\ ,q)$$

with $x \simeq |\,x_q>$ and x_q is logically *a priori* associated with some y_q via the function ϕ_m. On the assertion that x is associated to some y via the function ϕ_m, the matrix element (m,n,*,q) becomes a measure of the strength of this hypothesis. Therefore, to obtain the *optimum association*, we compute

$$\max_{m,n,q}\ (m,\ n,\ *,\ q)$$

and obtain the corresponding ϕ_m, whence we obtain the most logical association y via

$$M_r\,|\,\phi_m> = y$$

One could extend this notion of logical substitutability and reorganize the association space accordingly.

This is carried out next in our three-point memory system model as indicated below.

$$M_p^3 = \sum_{klm}\ \sum_{p}\ P\ \{\ (-1)^p\ \alpha_{klm}\,|\,\phi_k(1)\ \phi_l(2)\ \phi_m(3)>\}\ <x_k(1)\ x_l(2)\ x_m(3)\,|$$

where, P is a permutation operator with a parity p,

$$P = \begin{vmatrix} 1 & 2 & 3 \\ k & l & m \end{vmatrix}$$

Note that the right hand side is summed over all permutations. The matrix elements are then

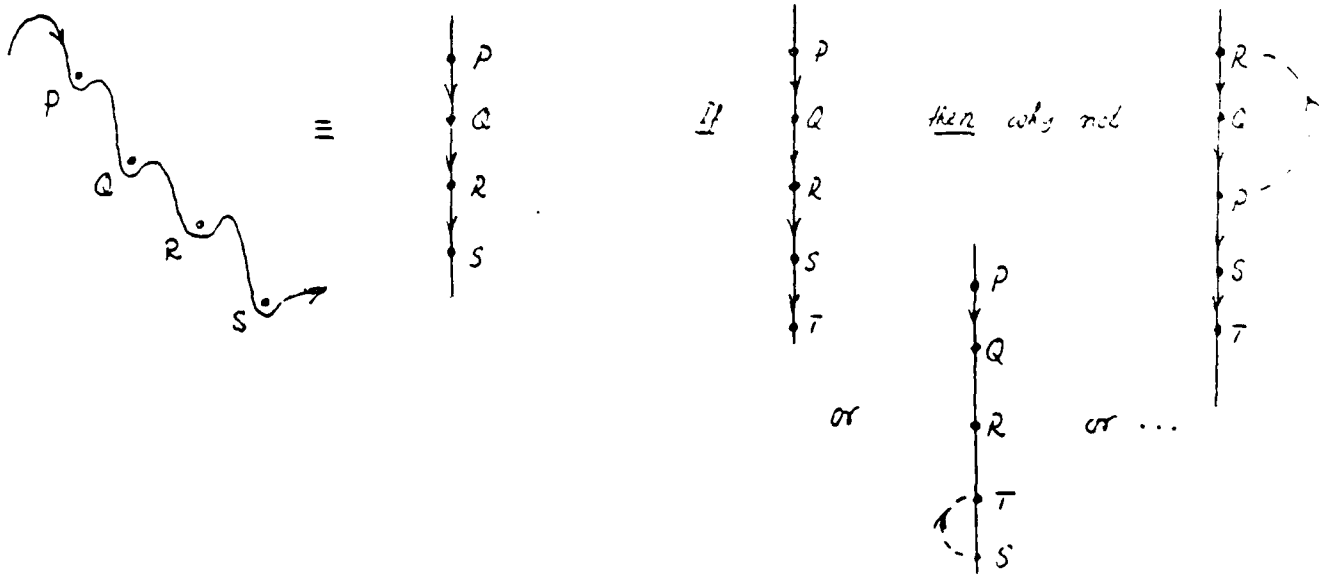
$$<\phi_r(1)\ \phi_s(2)\ \phi_t(3)\,|\,M_p^3\,|\,x_k(1)\ x_l(2)\ x_m(3)> = \sum_p P\ \{\ (-1)^p\ \alpha_{rst}\ a_{rk}\ a_{sl}\ a_{tm}\ \}$$

which then works out to

$$(r, s, t, k, l, m) = \alpha_{rst}\ a_{rk}a_{sl}a_{tm} - \alpha_{srt}\ a_{sk}a_{rl}a_{tm} - \alpha_{rts}\ a_{rk}a_{tl}a_{sm}$$
$$- \alpha_{tsr}\ a_{tk}a_{sl}a_{rm} + \alpha_{str}\ a_{sk}a_{tl}a_{rm} + \alpha_{trs}\ a_{tk}a_{rl}a_{sm}$$

where (r,s,t,k,l,m) is the matrix element in question. Note that the substitutability rationale could be advanced in the same way as we articulated it earlier in the two point memory system. If the string x_k is found similar to x_l and x_m then the strength of the hypothesis concerned with the *a priori* association of x_k with ϕ_k ought to be adjusted in view of the likelihood of false memory assignments via ϕ_k, ϕ_l, and ϕ_m. In our model, we propose that with respect to the original configuration $\phi_k(1)\ \phi_l(2)\ \phi_m(3)$ every other assignment of the form $\phi_{k_1}(1)\ \phi_{k_2}(2)\ \phi_{k_3}(3)$, where k_1, k_2, k_3 is some permutation of the indices k,l,m is either an *inhibitory* memory association (reduces the strength of our initial hypothesis) or a *contributory* memory association (advances the strength of our initial hypothesis) depending on the number of transpositions required to transform {k,l,m} to $\{k_1, k_2, k_3\}$. That is, if there are q such transpositions required to bring the string {k,l,m} into $\{k_1, k_2, k_3\}$ then the parity of the new memory assignment is $(-1)^q$ as though every single transposition is tantamount to an adjustment contrary to the direction of the original assignment.

We consider the following diagram to illustrate the point that our model suggests.



Assume that on the *energy "hill"*, at the consecutive local minima, the distinct patterns are stored as shown by the points P, Q, R, etc. Suppose, also, that these patterns are so close to each other that sometimes instead of retrieving the pattern Q from the storage we recall some pattern R. It is pointless to suggest that we sharpen our pattern specification more *tightly* with a view to minimizing pattern misclassification to some

tolerable level. Accordingly, we approach the problem in the following way. Indeed, on some cue, let us assume, it appears that the most probable recall is, say, pattern Q. However, what if we had the patterns stored in the order say R Q P S T ... or P Q R T S .. instead of the indicated order of storage on the hill P Q R S T ... Would that make any difference though? Would we still return the pattern R when we were supposed to return Q, instead? Surely, these alternative orders are also probable with some nonzero probabilities simply because the patterns (R and P) in the first suggested list, and the patterns (T and S) in the second list are conceivably *interchangeable* to the extent these few similar patterns are concerned. But, then, why restrict ourselves to single-transposition lists? Surely, if P Q R S T ... is a feasible list, then the list obtained by double transpositions such as (P R) (P T) {P Q R S T ... } = {T Q P S R ... } is also a probable. However, we make a somewhat qualified claim now. If the storage order in the first approximation on the "hill" seems to be the list P Q R S T ... coming down the *hill*, then a storage order implying a p-transposition on the original order ought to be less probable than the one with a q-transposition, if q < p. In other words, in our scheme of approach, the one point memory model is the most optimum first-order AM/CAM model we could think of. Its improvement implies a model in which interactions among patterns cannot be ignored any more. The two-points memory system, and then the three-points memory systems are then the second and the third-order approximations, as indicated, with more and more pattern-interactions taken into account.

Therefore, one could suggest a plausible model in which we carry out storage and retrieval of patterns as follows. We assume that AM/CAM memory need not always remain static. We, instead, suggest that such a memory should be dynamically restructured based on how successful one continues to be with the process of recall. We assume the memory system to be in state γ, that is its current memory is so densely packed that at time t it is a γ-point memory at some level p (p is an integer less than or equal to γ) as indicated below

$$\mathbf{M}_p^{\gamma} = \sum_{klm\ldots q} \sum_{p} \mathbf{P} \{(-1)^p \alpha_{klm\ldots q} \mid \phi_k(1)\phi_l(2)\phi_m(3)\cdots\phi_q(n_q)> \} \\ <x_k(1)x_l(2)x_m(3)\cdots x_q(n_q)\mid$$

If recall of similar patterns, in this memory, gives us a hit ratio less than some threshold parameter p_γ then we should *restructure* the memory as a $\mathbf{M}_{p+1}^{\gamma}$ element, and continue using it. This, probably, is what does tend to happen in using *biological memory*. Whenever we tend to face difficulties in recalling stored information (with a presumed similarity with other information strings), perhaps, even consciously at times, we restructure our memory elements focusing on other nontrivial features of the patterns not considered before.

Acordingly, one could approach the storage/retrieval policy as follows. Assume that the system is at state γ, i.e. its memory is at most $\mathbf{M}_{\gamma}^{\gamma}$. Given an incoming pattern $\overline{x}$, the system tries to return a vector $\overline{y}$ using the simplest memory $\mathbf{M}_1^{\gamma}$ in which interactions among the stored patterns are ignored. If, within some threshold parameter δ_1^{γ}, it finds that it could, instead, return any one of the vectors in the set $S_1^{\gamma} = \{ z_1^1, z_2^1, \ldots, z_q^1 \}$ then it considers its memory to be logically the $\mathbf{M}_2^{\gamma}$ and attempts to return $\overline{y}$ with less uncertainty. If, however, it finds that, within some threshold parameter δ_2^{γ}, it could, instead,

return any one of the vectors in the set $S_2^2 = \{z_1^2, z_2^2, \ldots, z_r^2\}$ then it considers its memory organized as M_3^3 and continues with the process. We assume that, while the system is at state γ, the memory could be allowed to evolve sequentially at most to its highest level M_γ^γ.

What if, the system fails to resolve an input cue vector even at the memory level M_p^p? Then, we could reject that pattern as irresolvable and continue with a fresh input. The system could be designed to migrate from the state γ to the state $\gamma+1$ if the number of rejects in the irresolvable class mounts up rapidly. Otherwise, we leave it at the state γ, and on each fresh cue $\vec{x}$ we let the system work through $M_1^1 \rightarrow M_2^2 \rightarrow M_3^3 \rightarrow \ldots \rightarrow M_\gamma^\gamma$ as need be.

References

1. Abu-Mostafa Y. S. and Jacques J. S. "Information Capacity of the Hopfield Model", IEEE Trans. Information Theory, IT-31, pp 461-464, July 1985.

2. Bak C. S. and Little M. J.: "Memory Capacity of Artificial Neural Networks with High Order Node Connections", Proc. IEEE Intl. Conf. Neural Networks, I. pp I207-I216, July 1988.

3. Chiueh T. and Goodman R. M.: "High Capacity Exponential Associative Memories", Proc. IEEE Intl. Conf. Neural Networks, 1, pp I153 - I160, July 1988.

4. Farhat N. H. *et al.* "Optical Implementation of the Hopfield Model", Applied Optics, 24, pp 1469-1475, May 1985.

5. Fuchs A. and Haken H.: "Pattern Recognition and Associative Memory as Dynamical Processes in Nonlinear Systems", Proc. IEEE Intl. Conf. Neural Networks, I, pp I217 - I224, July 1988.

6. Hinton G. E. and Anderson J. A. *Parallel Model of Associative Memory.* Lawrence Erlbaum Associates, Pub. Hillsdale, New Jersey, 1981.

7. Hopfield J. J. "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," Proc. Natl. Acad. Sci. USA, 79, pp 2554-2558, April 1982.

8. Kohonen T. "Correlation Matrix Memories", IEEE Trans. Computers, C-21, pp 353-359, April 1972.

9. Kohonen, T. *Associative Memory.* Springer-Verlag, New York 1977.

10. Lee Y. C. *et al*: "Machine Learning using a Higher order Correlation Network", Physica, 22D, pp 276-306, 1986.

11. McEliece R. J. *et al*, "The Capacity of Hopfield Associative Memory", IEEE Trans. Information Theory, IT-33, pp 461-482, July 1987.

12. Sussman H. J. "On the Number of Memories that can be Perfectly Stored in a Neural Net with Hebb Weights", IEEE Trans. Information Theory, IT-35, pp 174-1178, January 1989.

13. Oja E. and Kohonen T. : "The Subspace learning Algorithm as a Formalism for Pattern Recognition and Neural Networks", Proc. IEEE. Intl. Conf. Neural Networks, I, pp I277-I284, July 1988.

Models of Adaptive Neural-net based Pattern Classifiers/Recognizers

A neural network based system implemented as a collection of elemental processors working cooperatively in a functional setting in specific problem domains like controlling an unknown dynamical system autonomously is ultimately a learning entity that evolves through some form of self-organization. Learning is mediated via recurrent sampling on an open event domain based on the paradigm of *"learning by examples"*, whereby the pairwise connections among the "neurons", the synaptic weights, asymptotically tend to approach a globally stable equilibrium state on the assumption that the relevant training set itself remains stable during the learning phase. In this framework, an ideal neural network type system is functionally equivalent to an adaptive pattern recognizer whether it is designed to function as a front-end image-compressor system via an appropriate vector quantization of images, or as a system that yields optimum or almost optimum tours of Traveling Salesman problems. It is in this regard that neural network models have had been most extensively studied – as adaptive pattern classifier/recognizers, as systems that could autonomously learn and function in an unknown problem environment.

Accordingly, it is this specific area of clustering and pattern recognition in which major research thrust continues today (4) in one form or another. The problem here is primarily two fold: (a) identification of an appropriate learning (or mapping) algorithm, and (b), given (a), identification of an implementable architecture that reflects a functionally parallel distributed system and support of such a system functionally at a high-performance level. A neural network type system emerges as a powerful high-performance machine because it is inherently concurrent at instruction and at task levels; any compromise with this specific feature would surely lead to a precipitous drop in its performance. Given this imperative one could alternatively approach the problem from the opposite angle: Obtain an appropriate learning algorithm given that the processing architecture must be parallel-distributed with a maximum exploitation of system level concurrency.

The question of validity of neural network models is no longer the issue. The pertinent issue now is how to implement what need be implemented, i.e. the realization of a feasible and a promising learning algorithm, and the attendant system level architecture. In this paper, the framework is set on the assumption that learning categories of patterns by an unsupervised collection of automata in a distributed computation mode is where the research interest is focused. Accordingly, this is the area we concentrate on this paper.

Patterns over Feature space

Objects of interest, through appropriate feature extraction and encoding, are conveniently mapped onto patterns with binary-valued or analog features. Inversely, patterns over feature vectors could be made associative with the corresponding objects which constitutes the notion of equivalence classes in the following sense

- A pattern $x_i = (x_{1i}, x_{2i}, \ldots, x_{vi})^T$

is in class c_λ if the corresponding class-exemplar $\bar{x}_\lambda$ is
near to x_i within a tolerance distance (or an uncertainty) of δ_λ, i.e.

$$d(x_i, \bar{x}_\lambda\} \leq \delta_\lambda$$
where,
$$\lambda = \text{index } (\min_{\mu} \{d(x_i, x_\mu)\})$$

where d(.,.) is some suitable distance measure on the metric spanned by the pattern vectors.

In an unsupervised environment, the emerging class-exemplars or the cluster prototypes in a multiclass domain could be made to obey some additional constrains. Assuming that we do not *a priori* know their distributions except the requirement that their mutual separation must be at least as large as some threshold parameter, one could require that

$$d(\bar{x}_\lambda, \bar{x}_\mu) \geq \rho_{\min}$$

The idea is that at some distance or lower a pair of clusters may lose their individual distinctiveness so much so that one could merge the two to form a single cluster. Precisely how $\rho_{\min}$ ought to be stated is a debatable issue, but, it could be made related to the inter-cluster distance (ICD) distribution which we may conjecture to be a normal distribution with a mean $\bar{\rho}$ and a variance σ_ρ^2. Accordingly, we could let

$$\rho_{\min} = \bar{\rho} + p\, \sigma_\rho,$$

where p is some number around 1.0, and

σ_ρ is the variance of the ICD-distribution $N(\bar{\rho}, \sigma_\rho^2)$, assumed to be small. The problem is, that we usually do not know this distribution *a priori*. For an unknown pattern domain, it may not even be possible to predict *a priori* in how many distinct pattern classes should the pattern space be partitioned. All we know is that more separated the equivalence classes are more accurate would one be in ascertaining that a specific pattern belongs to a specific class.

Ideally, clusters ought to be well separated on the feature plane where the patterns are recognized. This is desired by a high value of $\bar{\rho}$. This could be achieved by designing a feature space in which small differences at individual pattern levels could be captured. This is a design issue, an encoding problem which could be separately tackled to an arbitrary degree of performance. This issue would be addressed in our subsequent papers. However, there is another problem, which we should look into. This arises in situations where the order of input matters. In a functional system, we do want the pattern-classes to be tight, i.e. we want the *intra-cluster distance* measures to be as small as possible so that we could have *crisp* clusters and retain them as such. This means that the variance measures on the exemplars or the average coefficient of variation of an exemplar ought to be small. But then, is this, or should it be ever under our control? The pattern-sampling that takes place during the training period may include event instances so noisy and incomplete that

for all practical purpose they could be considered garbage. Should we let the system be trained with such feature vectors? One could suggest that if on inclusion of such a feature vector the class prototype is perturbed more than one could tolerate then one could abandon the vector altogether. However, this means that to be reasonable one should make the level of tolerance function of number of patterns already accommodated within the class.

Some Neural Net Classifier Models

In this section, we conjecture some models which are obvious extrapolation of some neural-net based models already advocated. One could consider a known classifier model and expand it to suit a specific architecture, in which case one starts with the advantage that at least results up to the preextrapolated modeling state is known and could be considered as a reference basis for further studies. However, we have some problems in this regard. It is possible that

- without an extensive simulation and actual real case studies, one may not be in a position to confidently use these extrapolated models, even though, they do appear fairly plausible at this stage,

- no neural net model has been so extensively studied to understand the extent to which the model is applicable under realistic noise and measurement oriented issues,

- no research result is available to indicate to what degree the classification/recognition problem, in the domain of neural net paradigm, is sensitive to pattern symmetries particularly when one deliberately expands the feature space by systematic topological operations such as folding, contraction, etc. Note that such topological operations do not, in general, reduce the inherent entropy measures of patterns studied, but it could provide some redundancy on the metric space which could be exploited. It is not necessarily obvious that such a topological restructuring is always possible or even desirable for feature enhancement.

Accordingly, we propose the following. Of course, this has to be thoroughly tested, particularly with highly symmetric patterns and with those contaminated with noise, but we could extend the feature space from an orthogonal v-dimensional feature space to a ν-dimensional general feature space contemplated as follows. Given a pattern on the original vector space as a tuple $\{x_i\}$, we consider it equivalent to the nonorthogonal expansion in the following form

Original pattern $P = \{x_i\}$
Equivalent pattern on the extended space, $P_{eqiuv} = \{x_i, x_i x_j, i \neq j, x_i x_j x_k, i \neq j \neq k, \ldots\}$.

For classification/recognition purpose, we assume $P_{equiv} \simeq P$. The additional redundancy

introduced by this artificial encoding would not be detrimental; on the contrary, it would assist us to discriminate patterns successfully in a parallel-distributed processing scheme.

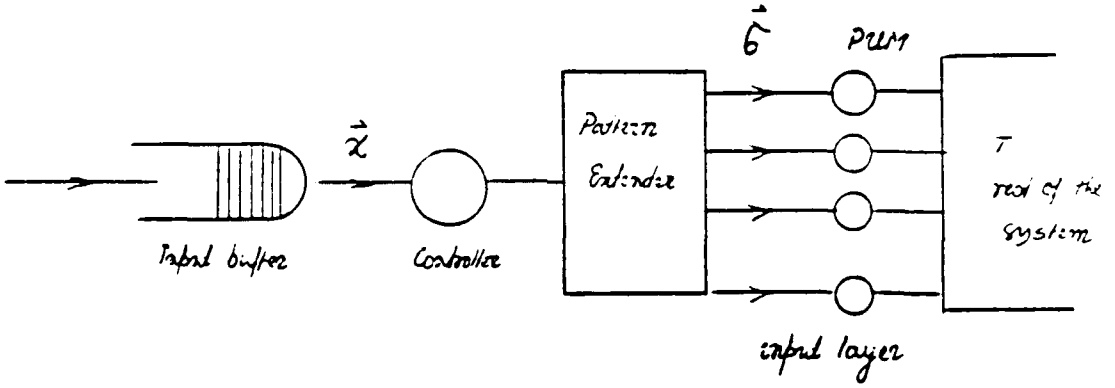
A Localized-Distributed ART1 Scheme:

In this scheme, the basic ART1 model of Carpenter & Grossberg (1,5) that deals with binary feature vectors is considered as the reference system for developing a workable localized-distributed model for feature recognition/classification. This is outlined as follows.

(a) Given feature vectors $\vec{x}$ on the original v-dimensional basis, we obtain, for each pattern, the extended, but equivalent, feature vector $\vec{\sigma}$ as indicated earlier. The entire σ-pattern is input to a group of processing unit modules each of which comprises a number of neuron-type processors. Thus, logically, the pattern vector $\vec{\sigma}$ is captured by, say, N PUM (processing unit module), each PUM comprising, say, n neuron-type simple processor elements receives, at time t, a specific partition of the $\vec{\sigma}$ pattern for processing. Thus, the Ith PUM would receive, at time t, the following ordered input string for processing

$$\vec{\sigma}_I(t) = \{\sigma^t{}_{I1}, \sigma^t{}_{I2}, \ldots, \sigma^t{}_{In}\}$$

This is as shown below.



Input Preparation Stage

(b) For each Ith PUM, we have the local group vigilance parameter ρ_I which is a variable. For each pattern class or exemplar j, similarly, we have a separation measure π_j reflecting the minimum tolerable distance the other classes must be at given the pattern class j. Initially,

$$\rho_I = \rho^* + [1 - e^{-f(\omega)}]\, u(\rho_c - \rho_0, 0)$$

and

$$\pi_j \geq 0$$

where u(x,y) is a uniformly distributed random variable between x and y, both inclusive, and f(m) is some appropriate monotonically increasing function on the PUM size m, the number of elemental neuron type processors it contains. Also,

$$\rho^* = \min_I \{\rho_I\}, \text{ and } \rho_c = \max_I \{\rho_I\}$$

(c) each PUM, working independently from each other, attempts to identify the best exemplar its input pattern corresponds to. We assume that there are, $\mu + 1$ distinct classes to choose from. One of these, the refuse class, would correspond to those patterns which are too fuzzy to belong to any of the other μ classes. The exemplar weights are stored in the b-matrix. The dot products of its input pattern with the exemplar patterns are computed at this level as

$$\mu^I_j = \sum_i b^I_{ji}\, \sigma_{Ii}$$

where, the index i is over all neuron-type processors comprising the Ith PUM, and the index j points to a specific class or pattern exemplar.

(d) The tentative cluster $I(j^*)$ for the Ith PUM conglomerate is obtained as

$$I(j^*) = \text{index } (\max_j \{\mu^I_j\})$$

If $\mu^I_{j^*}$ is less than δ_{j^*} then the pattern goes to refuse class.

(e) For each PUM, we obtain α_I as given by

$$\alpha_I = \frac{\sum_I t_{I(j^*)i}\, \sigma_{Ii}}{\sum_I \sigma_{Ii}}$$

(f) If $\alpha_I \leq \rho_I$, then we deactivate the suggested class $I(j^*)$, as shown by Carpenter & Grossberg, and go back to (c). Otherwise, at the earliest possible time, we post $I(j^*)$ at the next level up with its $(\alpha_I - \rho_I)$ value.

(g) We next compute the frequency of occurrence $m(j^*)$ of the exemplar j^* as the indicated pattern choice at the subpattern levels, and obtain

$$\beta(j^*) = m(I(j^*)) \sum_I \frac{(\alpha_I - \rho_I)\ \text{Thr}\ (\alpha_I - \rho_I)}{\rho_I}$$

where, Thr(x) = 1 if x > 0.0, otherwise, 0.

(h) The most suggested cluster center is j^{**} where

$$j^{**} = \text{index}\ (\max_{j^*} \{\beta(j^*)\}\ )$$

(i) Synaptic weights (both top-down and the bottom-up weights) are updated if $\beta(j^{**}) > \gamma$, some threshold parameter. Then, referring to time instances by the label l, we have,

$$t^I_{j^{**}i}(l+1) = t^I_{j^{**}i}(l)\ \sigma_{\Pi}$$

$$b^I_{j^{**}i}(I+1) = \frac{t^I_{j^{**}i}\ (l+1)}{0.5 + \sum_i t^I_{j^{**}i}\ (l)\ \sigma_{\Pi}}$$

(j) If the weights are updated via (i) then on all PUM nodes I with $I(j^{**})$ as its choice, we update

$$\rho_I(l+1) = \text{avg}_I\ \{\rho_I\ (l)\ \}$$

else, we update them as

$$\rho_I\ (l+1) = \max_I\ \{\rho^*\ ,\ \rho_I(l)\}$$

The algorithm as outlined here is an extension of the basic approach articulated as ART1 algorithm of Carpenter and Grossberg (3). The idea here is to discover the global cluster structure via cluster formation at local level. If any such algorithm could be outlined, then it is of immense help to us since any pattern structure could be, in principle, partitioned into a number of parallel processor unit modules each of which would then be required to identify locally at its level whatever distinguishing pattern it can see.

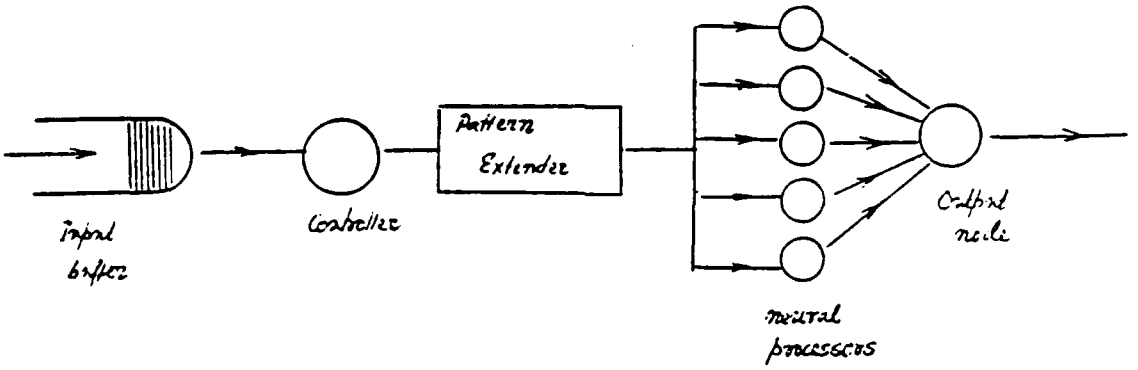
A Bland-net Classifier:

This approach, though lacks depth, is useful as an alternative cluster determination procedure. In spite of the fact that it is presented as a centralized scheme, it could be easily restructured as a localized-distributed scheme as outlined in the previous section with the ART1 type algorithm. Its basic architecture is a single-level, or a single-layer multiple neuron-type processors processing the input string, and a decision-unit, sitting at its top working on the output from the neurons below. This is outlined below.

(a) As in localized-distributed ART1 scheme, extend the input feature vector $\vec{x}$ to a feature vector $\vec{\sigma}$. The input vector is now the ordered string

$$\vec{\sigma}(t) = \{\sigma^t{}_1, \sigma^t{}_2, \ldots, \sigma^t{}_n\}$$

(b) Assign random weights to the synaptic elements b_{ji}. Note that the elements $b_{ji} \in [0,1]$.



A Bland-net System

(c) On an input pattern to be classified, compute pattern-pattern class and pattern class-pattern class distances over the Mahalnobis metric with a positive value for the λ parameter

$$D_j = \left| \sum_i (b_{ji} - \sigma_i)^\lambda \right|^{\frac{1}{\lambda}}$$

$$Q_j = \min_k \sum_i (b_{ji} - b_{ki})^2$$

(d) Compute $j^* = \text{index}\,(\min_j \{D_j\})$. If $D_{j^*} < \delta$, then obtain

$j^* = \text{index}\,(\min_j \{\frac{D_j}{Q_j}\})$. Pattern σ is in class j^*.

(e) Update synaptic weights b_{j^*i} if the last pattern is included in this class as

$$b_{j^*i}\,(n_{j^*} + 1) = \frac{n_{j^*}}{n_{j^*} + 1} b_{j^*i} + \frac{1}{n_{j^*} + 1} \sigma_i$$

where n_{j^*} is the number of patterns already accommodated in the cluster j^*.

Training a system to the eventual task of correct pattern-class identification on a pattern space may be order-sensitive in some cases. This, of course, depends upon the algorithm used to identify clusters, and the way the training patterns are introduced at the input and processed by the system subsequently. If the system at the task level is essentially sequential in the sense that the individual patterns are sequentially admitted and processed one at a time against the already acquired knowledge, tentative though may be, of likely class-distribution, which, in turn, is expected to be incrementally adjusted given the current new input, we could get an order sensitive distribution eventually. Ideally, the input patterns, in its entirity should be concurrently processed as one single whole without any prior reference to any distribution, so that after a given numb number of cycles, we would obtain a stable, crisp, unbrittle distribution of the sampling population significantly close to the population distribution in the universe of discourse. But, if the patterns arriving at an input port are only sequentially admitted as units of a single time-dependent or a time-varying input stream, we cannot arbitrarily keep them on hold in some temporary buffers to be released at some convenient time for concurrent processing in bulk, unless the input rate is very high compared to processing rate.

To understand the overall dimension of the problem in this regard, we consider some simple processing models. Suppose, the patterns arrive at a rate of λ_p patterns/sec at an input port obeying a Poisson distribution as follows. The probability that the number of arriving input patterns during an arbitrary interval of length t units is n is given by

$$\text{p(number of inputs = n within t)} = e^{-\lambda_p t} \frac{(\lambda_p t)^n}{n!}$$

Suppose, in the first model, we have all the incoming patterns temporarily queued in a buffer (infinitely long) till they are serviced one at a time in a FIFO discipline by an appropriate classifier (using, say, ART1 algorithm of Carpenter & Grossberg (3.5)) in which the average individual pattern processing time is $1/\mu_c$ sec. We assume that the classification process is exponentially distributed, that is the probability distribution of the classification/recognition time is given by the

$$\begin{aligned} f(s) &= \mu_c\, e^{-\mu_c s}, \qquad s \geq 0 \\ &= 0, \qquad \text{otherwise} \end{aligned}$$

In this case, the relevant system and pattern traffic aggregate measures emerge as follows (2).

$$L^0, \text{ the average buffer occupancy} = \frac{\lambda_p}{\mu_c - \lambda_p}$$

$$W^0, \text{ the average pattern-residence time in the system} = \frac{1}{\mu_c - \lambda_p}$$

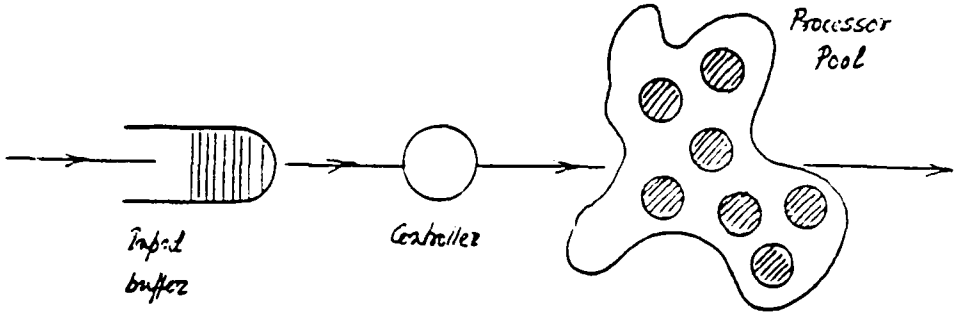
$$W^{q0}, \text{ the average pattern time in the buffer} = \frac{\lambda_p}{\mu_c(\mu_c - \lambda_p)}$$

We should strive for a system in which L^0, W^0, and W^{q0} are all small.

We could improve the system somewhat if we process the patterns concurrently by a set of multiple classifiers. We assume that they are all equal in performance, i.e. capable of classifying a single pattern in $1/\mu_c$ time on the average. There are at least two distinct ways to process the patterns now.

Processor Pool Model A:

We assume that the system, in this case, comprises k identical processors in a processor pool to process incoming patterns in FIFO mode service. In this case, all the incoming patterns wait in a single queue served by k identical processors in the M/M/k service mode. Whenever one of the k-machines becomes idle, it picks up the first pattern it finds at the input queue and attempts to classify it. The system organization is indicated below.



A Processor Pool based Classifier Model

Given that the arrival rate of the patterns into the system is still λ_p patterns sec., and the service rate at any of the k given classifiers is also Poisson with a mean rate of μ_c, and that $\lambda_p < k\ \mu_c$, the equilibrium probability p_0 that the system is idle is given by

$$p_0 = [\sum_{i=0}^{k-1} (\frac{\lambda_p}{\mu_c})^i + \frac{1}{k!} (\frac{\lambda_p}{\mu_c})^k \frac{k\mu_c}{k\mu_c - \lambda}]^{-1}$$

and, the equilibrium probability p_i that the system has i patterns in it is given by

$$p_i = \frac{1}{i!}(\frac{\lambda_p}{\mu_c})^i p_0, \qquad \text{for } i \leq k$$
$$= \frac{1}{k!}(\frac{\lambda_p}{k\mu_c})^i k^k p_0, \qquad \text{for } i > k$$

in terms of which the system aggregates could be computed as

$$L_q = \frac{p_0 (\frac{\lambda_p}{\mu_c})^c (\frac{\lambda_p}{k\mu_c})}{k! [1 - (\frac{\lambda_p}{k\mu_c})]^2}$$

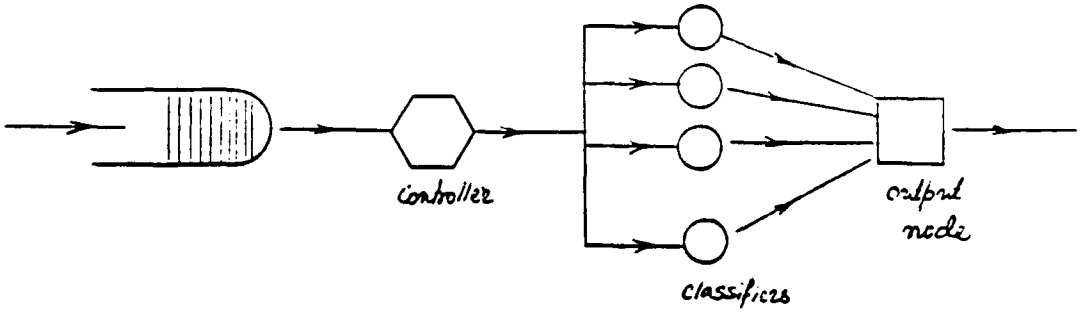
where, L_q is the average number of patterns in the processing queue waiting for one of the k processors to process them, and,

$$W^A = \frac{L_q}{\lambda_p} + \frac{1}{\mu_c}, \quad \text{and}$$
$$L^A = \lambda_p W^A$$

where W^A is the average residence time per pattern in the system, and L^A, respectively is the system occupancy rate in the model A.

k-Parallel Stream model B:

In this case, the input stream is split into k streams each of which is then looked after by a single processor with a processing power, as before, at a rate of completing on the average of one pattern in every μ_c second. This is schemetically as shown below.



Parallel Stream or SIMD based design

At each arriving pattern stream, we assume, the interarrival time between any two consecutive patterns, on the average, would be exponentially distributed with a mean of $\frac{k}{\lambda_p}$ since upon being available for processing it would be sent down to one of the processor queue with a probability of 1/k. In this case, the system aggregates are as follows

$$L^B = \frac{\lambda_p}{k\mu_c - \lambda_p}$$

with the average residence time per pattern as

$$W^B = \frac{k}{k\mu_c - \lambda_p}$$

In either case, whether we provide a bank of multiple servers for a single queue pattern traffic or a multiple port service with each input port being serviced by a single processor, the net result is the reduction of residence times, since both $\frac{W^A}{W^0}$ and $\frac{W^B}{W^0}$ are less

than 1, for k >1.

A Quasi-Array Processor Model:

Here, we assume the system to be in one of the two mutually exclusive states. We assume that it is either at a learning state A whereupon it trains itself incrementally with a stream of incoming feature vectors on the assumption that these vectors do constitute a set of legitimate patterns from the population of patterns in a statistical sense, or it is in state B where its learning has stabilized to the extent of knowing precisely in how many pattern classes the pattern domain is to be partitioned, and also, to a large extent, how the corresponding pattern exemplars look like.

Assuming the system to be in state A, that is in the learning state, we let the incoming feature vectors all collated in a time ordered sequence at the input buffer B serviced by a controller. The controller, periodically, picks up k feature vectors from the buffer and gets them shuffled at a Permutation block P such that the input string of vectors $\{\vec{x}(t), \vec{x}(t+1), ..., \vec{x}(t+k-1)\}$ is changed to a random sequence of vectors of the form $\{\vec{x}(t_k), \vec{x}(t_l), ..., \vec{x}(t_m)\}$ at the output of P. The controller then dispatches these vectors to the classifiers at a rate of one vector per classifier per dispatch event. The classifier bank comprises k independent classifying units. Each classifier C_i upon receiving the vector $\vec{x}(t_j)$ attempts to learn from it using one of the algorithm outlined. After learning/classifying m consecutive patterns, each classifier C_i sends its exemplar profile map to its output node O_i where the output exemplar-profile corresponds to the pattern-class distribution for the last m independent feature vector events. The output profile at O_i is a string of vectors along with their pattern counts i.e. the number of patterns it has accommodated in a given class, as shown below.

$$\psi_i\ (l) = \{(\vec{Z}^i{}_1, m^i{}_1), (\vec{Z}^i{}_2, m^i{}_2), ..., (\vec{Z}^i{}_h, m^i{}_h)\}$$

with $m^i{}_j$ = the cardinality of the class j given m patterns in total

indicating that at the end of m consecutive pattern processings the plausible partition of the feature-space is a h-tuple as shown above. The local output node O_i sends this feature map to the next level up at the node Y where all such k independent feature maps arising from below are consolidated with the consolidated feature map obtained at the last cycle.

If, on the other hand, the system is at state B, that is in the state of functioning as a pattern recognizer, the controller sends the input pattern to be recognized directly to the processor Y in an M/M/c processing mode assuming the feature vector arrival process at the input buffer is Poisson and the pattern recognition time at Y is exponentially distributed, respectively. The service times at the processors C_i are all assumed to be exponentially distributed.

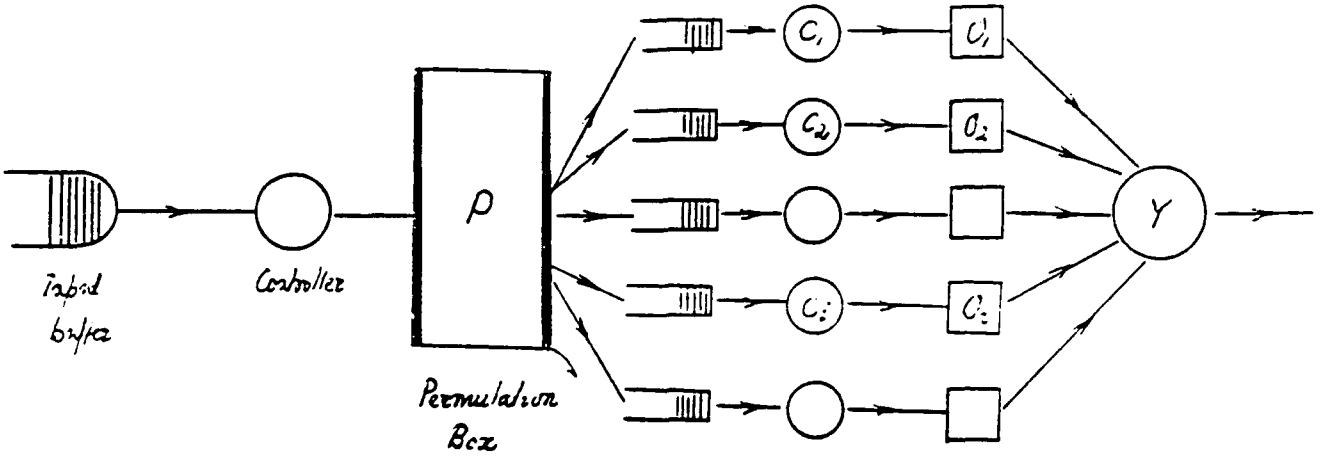
We assume that the switching over from the state A to state B takes place once the system is triggered by one or more of the following conditions. It is

- The input stream is punctuated by a time gap. After the time-out,

the system migrates into the state B.

• The class-exemplar profiles at Y over last q updates involve exemplar-pattern perturbations whose maximum is lower than some threshold parameter.

Note that at any sampling event time during or after training episodes, after some minimal time point t_0, the node Y has a specific exemplar profile. Even if one or more classifier type processors fault the system is globally fault-tolerant. The basic structure is indicated below.



Note that each elemental concurrent processor C_i could be, at some level, a feedback processor, as shown in the diagram, in the event the pattern-class corresponding to an input pattern requires further processing.

Consolidation of local distributions into a global pattern-class distribution

At the node Y, the global pattern-classification emerges asymptotically. That is, if the class-distribution profile at Y at time t=l is given by the distribution $\phi_Y(t=l)$, then $\lim_{t \to \infty} \phi_Y(t) = \phi_Y$, we assume, does exist and constitutes the desired class-exemplar profiles. We assume that at the end of some computational cycle l, the individual class-distribution profiles together with their local frequency of distribution $\{\psi_i(l), 1 \leq i \leq k\}$ are amassed from the output nodes $\{O_i\}$ below. All these are reconciled, at the end of the lth cycle, with the previously computed global class-distribution $\phi_Y(l\text{-}1)$ at Y as follows.

$$\phi_Y(1) = \psi_1(1)$$
$$\phi_Y(l) = (\phi_Y(l\text{-}1) \cup (\bigcup_i^k \psi_i(l)))$$

and, given two class-distribution profiles

$$\psi_i\ (l) = \{(\vec{Z}^i_1, m^i_1), (\vec{Z}^i_2, m^i_2), ..., (\vec{Z}^i_h, m^i_h)\}$$
$$\psi_j\ (l) = \{(\vec{Z}^j_1, m^j_1), (\vec{Z}^j_2, m^j_2), ..., (\vec{Z}^j_k, m^j_k)\}$$

we define the "union" operation of the above profiles as

$$\psi_i(l) \cup \psi_j(l) = \{ \left(\frac{m^i_q z^i_q + m^j_r z^j_r}{m^i_q + m^j_r} \right), (m^i_q + m^j_r) \mid d(z^i_q, z^j_r) \leq \delta \}$$

where, d(.,.) is the "distance" between the exemplar patterns in the argument. One could, if necessary, condition the union operation further by requiring that two classes may not be allowed to coalesce if the resultant class were to emerge as too big a class by size.

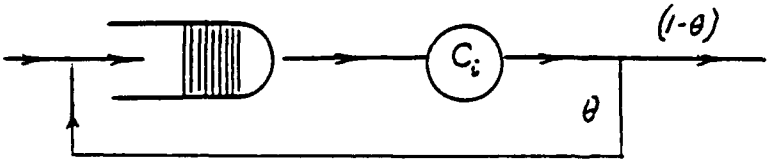
Performance Profile

The above model, as proposed, attempts to provide a "bias-free" architecture via concurrent multistream computation of the global class-distribution. If the patterns or rather the feature vectors, during the training period, arrive into the system at a Poisson rate of λ_p vectors/sec, each processing stream then receives input at a rate of $\frac{\lambda_p}{k}$ vectors/sec which gets processed with an average residence time of W sec. per pattern, where

$$W = \frac{1}{\mu_Y} + \frac{k}{k\mu_c - \lambda_p}$$

where, μ_Y is the average processing rate of feature vectors at the level Y.

Note that, in some cases, particularly in Carpenter/Grossberg-type algorithms, the internal processing by the elemental processor C_i may include feedback loop as shown below. This is due to the fact that sometimes a tentative class exemplar may not *resonate* at a specific class in which case one has to deactivate the class temporarily and search for an alternative choice by going back into the algorithm at least one more time.



An Internal Classifier with a Feedback Loop

Assuming that with a probability θ, a task returns to the end of the buffer for at least one more time of processing, and with a probability of (1 - θ) it departs the classifier, one

obtains, at equilibrium,

$$\frac{\lambda_p}{k} p_0 = (1-\theta)\ \mu_c\ p_1$$
$$(\frac{\lambda_p}{k} + (1-\theta)\mu_c)p_n = \frac{\lambda_p}{k} p_{n-1} + (1-\theta)\mu_c p_{n+1},\ n \geq 1$$

where p_n is the equilibrium or the steady-state probability that the subsystem at the C_i processor stream has n tasks in it (including one being processed, if any). The equilibrium population densities come out to be

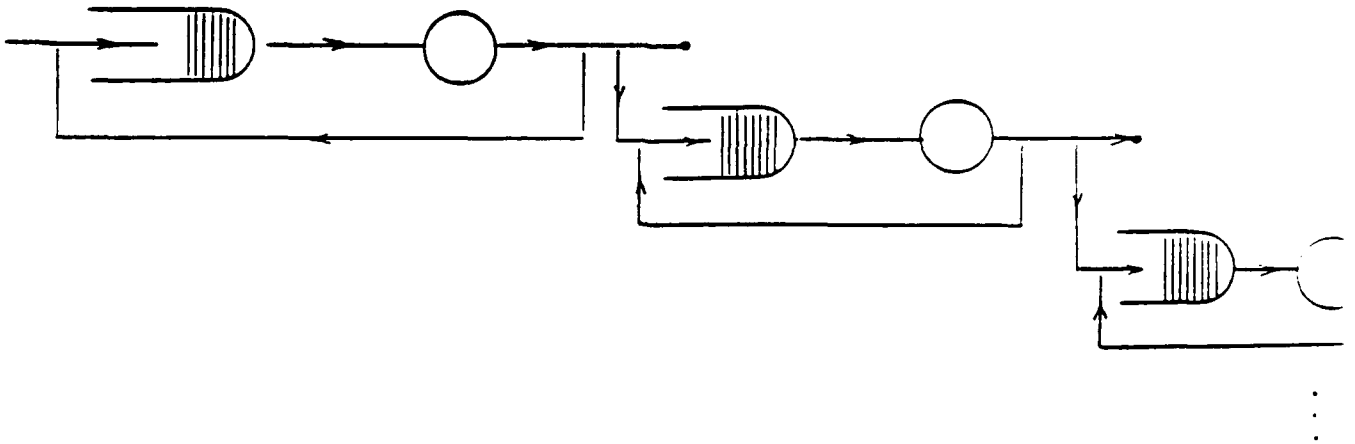
$$p_n = (\frac{\lambda_p}{k(1-\theta)\mu_c})^n\ (1 - \frac{\lambda_p}{k(1-\theta)\mu_c}),\ n \geq 0$$

and the steady state system occupancy rate L_i comes out as

$$L_i = \sum_0^\infty n p_n = \frac{\lambda_p}{k(1-\theta)\mu_c - \lambda_p}$$

while the expected response time of a task in this stream, W_i, comes out to be

$$W_i = \frac{L_i k}{\lambda_p}$$



A Multidepth Pipeline Classifier Processor

Furthermore, one could extend this architecture to a multidepth pipeline organization indicated below in which at a certain depth ξ either a feature vector is *processed* to the extent of knowing in which class it best belongs or it is not, in which case the pattern vector is passed onto the next level classifier working at a finer-grain resolution at a depth $(\xi + 1)$. Assuming that with a probability of ζ_ξ at a level ξ a feature-vector is referred to the next level processor, and with a probability of θ_ξ it loops back for further processing at the

level ξ, and assuming that, for all practical purpose, the system is decomposable as multiple M/M/c subsystems, working in tandem, we have the effective pattern arrival rates in each subsystem at the depth ξ as, at equilibrium,

$$\hat{\lambda}_\xi = \varsigma_\xi \frac{\lambda_p}{k(1-\theta_\xi)}$$

in terms of which, the expected residence time of a pattern task, at each pipeline section comes out to be

$$W_\xi = \frac{1}{(\mu_{c_\xi} - \hat{\lambda}_\xi)}$$

Note that an infinite variety of parallel distributed architecture for neural-based network systems could be proposed. In almost all cases, multiple neuron-type processors are best suited to provide SIMD type architecture at some computational level. Organization of machines based on SIMD type primitives as indicated above yield substantial performance.

References:

1. Lippman, R. P. "An Introduction to Computing with Neural Nets", *IEEE ASSP Magazine, April 4, 1987.*

2. Ross, Sheldon. *An Introduction to Probability Models.* Prentice-Hall Publishers, 1980.

3. Carpenter G. A. and S. Grossberg. "ART2: Self-organization of stable category recognition codes for analog input patterns"s."*Applied Optics, 26, pp 4919-4930, 1987*

4. Duda R. O. and P. E. Hart. *Pattern Classification and Scene Analysis*". Wiley, New York 1973.

5. Carpenter G. A. and S. Grossberg. "Category Learning and adaptive pattern recognition, a neural network model."*Proc. Third Army Conference on Applied Mathematics and Computing,* ARO Report 86-1, pp 37-56, 1985.

DISCRETE FOURIER TRANSFORMS ON HYPERCUBES

This paper presents a decomposition of the discrete Fourier transform matrix that is more explicit than what is found in the literature. It is shown that such a decomposition forms an important tool in mapping the transform computation onto a parallel architecture such as that provided by the hypercube topology. A basic scheme for power-of-a-prime length transform computations on the hypercube is discussed.

The use of the Chinese Remainder Theorem in pipelining the computation of the transforms, for lengths that are not a power of a prime, is also discussed.

1. A DFT matrix decomposition

Recall that the discrete Fourier transform Y of length n of a vector X of length n is defined by

$$Y = F\,X$$

where F is the square matrix of order n given by

$$F = [\, W^{kj} \,]$$

with $0 \le k, j \le n - 1$ and where W is the nth root of unity

$$W = e^{-2\pi i/n} = \cos(2\pi/n) - i \sin(2\pi/n),$$

with $i^2 = -1$.

It is claimed that if $n = 2^{\gamma}$, then the DFT matrix F is, up to a permutation matrix, the product of $\gamma = \log n$ sparse matrices

$$F \simeq F_0 F_1 F_2 \cdots F_{\gamma-2} F_{\gamma-1}$$

where each $F_{\gamma-i}$ is a block diagonal matrix

$$F_{\gamma-i} = \mathrm{diag}(F_{\gamma-i,0}\, F_{\gamma-i,1} \cdots F_{\gamma-i,2^{i-1}-1})$$

and the following properties hold

(i) each block $F_{\gamma-i,\alpha}$ is square of order $2^{(\gamma-i)+1} = n/2^{i-1}$ and is of the form

$$F_{\gamma-i,\alpha} = \begin{bmatrix} I & W^{r} I \\ I & W^{r+\frac{n}{2}} I \end{bmatrix}$$

and there are exactly 2^{i-1} blocks; I is the identity matrix of order $n/2^i$;

(ii) for each block, r is the nonnegative integer whose γ-bit binary expansion is the reversal of that of 2α.

The proof of this decomposition can be sketched as follows.

We write the components of Y and X as y[k] and x[j], for $0 \le k, j \le n - 1$, so that

$$y[k] = \sum_{j=0}^{n-1} x[j] W^{kj}$$

We then write the indices k and j in binary, using γ bits of course, i.e

$$k = k_{\gamma-1}k_{\gamma-2}k_{\gamma-3} \cdots k_2k_1k_0$$
$$j = j_{\gamma-1}j_{\gamma-2}j_{\gamma-3} \cdots j_2j_1j_0$$

In decimal,

$$k = 2^{\gamma-1}k_{\gamma-1} + 2^{\gamma-2}k_{\gamma-2} + 2k_1 + k_0$$
$$j = 2^{\gamma-1}j_{\gamma-1} + 2^{\gamma-2}j_{\gamma-2} + 2j_1 + j_0$$

Hence, with indices written in binary, we get

$$y[k] = y[k_{\gamma-1}k_{\gamma-2} \cdots k_1k_0]$$

$$x[j] = x[j_{\gamma-1}j_{\gamma-2} \cdots j_1j_0]$$

$$W^{kj} = W^{(2^{\gamma-1}k_{\gamma-1}+2^{\gamma-2}k_{\gamma-2}+\cdots+2k_1+k_0)(2^{\gamma-1}j_{\gamma-1}+2^{\gamma-2}j_{\gamma-2}+\cdots+2j_1+j_0)}$$

$$y[k_{\gamma-1}k_{\gamma-2} \cdots k_1k_0] = \sum_{j_0=0}^{1} \Big(\sum_{j_1=0}^{1} \big(\cdots \big(\sum_{j_{\gamma-2}=0}^{1} \big(\sum_{j_{\gamma-1}=0}^{1} x[j_{\gamma-1}j_{\gamma-2} \cdots j_1j_0] \cdot$$

$$W_{\gamma-1}) W_{\gamma-2}) \cdots) W_1) W_0$$

where

$$W_{\gamma-1} = W^{(2^{\gamma-1}k_{\gamma-1}+2^{\gamma-2}k_{\gamma-2}+\cdots+2k_1+k_0)2^{\gamma-1}j_{\gamma-1}}$$

$$W_{\gamma-2} = W^{(2^{\gamma-1}k_{\gamma-1}+2^{\gamma-2}k_{\gamma-2}+\cdots+2k_1+k_0)2^{\gamma-2}j_{\gamma-2}}$$

$$\vdots$$

$$W_1 = W^{(2^{\gamma-1}k_{\gamma-1}+2^{\gamma-2}k_{\gamma-2}+\cdots+2k_1+k_0)2j_1}$$

$$W_0 = W^{(2^{\gamma-1}k_{\gamma-1}+2^{\gamma-2}k_{\gamma-2}+\cdots+2k_1+k_0)j_0}$$

Since $W^{2\gamma} = W^n = 1$, we actually get

$$W_{\gamma-1} = W^{2^{\gamma-1}k_0 j_{\gamma-1}}$$

$$W_{\gamma-2} = W^{2^{\gamma-2}(2k_1+k_0)j_{\gamma-2}}$$

$$\vdots$$

$$W_1 = W^{(2^{\gamma-2}k_{\gamma-2}+\cdots+2k_1+k_0)2j_1}$$

$$W_0 = W^{(2^{\gamma-1}k_{\gamma-1}+2^{\gamma-2}k_{\gamma-2}+\cdots+2k_1+k_0)j_0}$$

The computation can then be carried out in stages, where at each stage we calculate an intermediate vector

$$X_{\gamma-i}$$

Note that in our notation, the first vector to be computed is $X_{\gamma-1}$ and the last is X_0.

From the expansion of

$$y[k] = y[k_{\gamma-1}k_{\gamma-2}\ldots k_1 k_0]$$

we see that the components of the vectors $X_{\gamma-i}$ and $X_{\gamma-i+1}$ are related by the equations

$$x_{\gamma-i}[k_0 \cdots k_{i-3}k_{i-2}k_{i-1}j_{\gamma-i-1}j_{\gamma-i-2} \cdots j_1 j_0] =$$

$$\sum_{j_{\gamma-i}=0}^{1} x_{\gamma-i+1}[k_0 \cdots k_{i-3}k_{i-2}j_{\gamma-i}j_{\gamma-i-1}j_{\gamma-i-2} \cdots j_1 j_0] \cdot$$

$$W^{2^{\gamma-i}(2^{i-1}k_{i-1}+2^{i-2}k_{i-2}+\ldots+k_0)j_{\gamma-i}}$$

$$= x_{\gamma-i+1}[k_0 \cdots k_{i-3}k_{i-2}\ 0\ j_{\gamma-i-1}j_{\gamma-i-2} \cdots j_1 j_0] \cdot$$

$$W^{2^{\gamma-i}(2^{i-1}k_{i-1}+2^{i-2}k_{i-2}+\ldots+k_0)\cdot 0} +$$

$$x_{\gamma-i+1}[k_0 \cdots k_{i-3}k_{i-2}\ 1\ j_{\gamma-i-1}j_{\gamma-i-2} \cdots j_1 j_0] \cdot$$

$$W^{2^{\gamma-i}(2^{i-1}k_{i-1}+2^{i-2}k_{i-2}+\ldots+k_0)\cdot 1}$$

$$= x_{\gamma-i+1}[k_0 \cdots k_{i-3}k_{i-2}\ 0\ j_{\gamma-i-1}j_{\gamma-i-2} \cdots j_1 j_0] +$$

$$x_{\gamma-i+1}[k_0 \cdots k_{i-3}k_{i-2}\ 1\ j_{\gamma-i-1}j_{\gamma-i-2} \cdots j_1 j_0] \cdot$$

$$W^{2^{\gamma-i}(2^{i-1}k_{i-1}+2^{i-2}k_{i-2}+\ldots+k_0)}$$

i.e we may write

$$X_{\gamma-i} = F_{\gamma-i} \cdot X_{\gamma-i+1}$$

for some appropriate matrix $F_{\gamma-i}$.

From the last expression above, we see that the matrix $F_{\gamma-i}$ can be blocked into 2^{i-1} submatrices. Each of the submatrices is determined by a unique value of the bits that are more significant than the (i-1)th bit, namely

$$k_0\, k_1 \ldots k_{i-3}\, k_{i-2}$$

We use these bits as our label α for each block, so the decimal value of α is

$$2^{i-2}k_0 + 2^{i-3}k_1 + \ldots + 2k_{i-3} + k_{i-2}$$

The size of each block $F_{\gamma-i,\alpha}$ is determined by a unique value of the bits that are less significant than those of the label α. We can see that there are γ-i+1 such bits, hence the size of $F_{\gamma-i,\alpha}$ is

$$2^{\gamma-i+1} = 2^{\gamma}/2^{i-1} = n/2^{i-1}$$

We next need to show that $F_{\gamma-i,\alpha}$ is given by

$$F_{\gamma-i,\alpha} = \begin{bmatrix} I & W^{r}I \\ I & W^{r+\frac{n}{2}}I \end{bmatrix}$$

For a fixed $\alpha = 2^{i-2}k_0 + 2^{i-3}k_1 + \ldots + 2k_{i-3} + k_{i-2}$, the two possible values for k_{i-1} yield the two halves of the matrix $F_{\gamma-i,\alpha}$.

Indeed when $k_{i-1} = 0$ then

$$x_{\gamma-i}[k_0 \cdots k_{i-3}k_{i-2}\ 0\ j_{\gamma-i-1}j_{\gamma-i-2}\cdots j_1j_0] =$$

$$x_{\gamma-i+1}[k_0 \cdots k_{i-3}k_{i-2}\ 0\ j_{\gamma-i-1}j_{\gamma-i-2}\cdots j_1j_0] +$$

$$x_{\gamma-i+1}[k_0 \cdots k_{i-3}k_{i-2}\ 1\ j_{\gamma-i-1}j_{\gamma-i-2}\cdots j_1j_0] \cdot$$

$$W^{2^{\gamma-i}(0+2^{i-2}k_{i-2}k_{i-3}+\cdots+k_0)}$$

Now the submatrices I and $W^{r}I$, where I is the identity matrix of order $n/2^{i}$ are evident, where r is given by

$$r = 2^{\gamma-i}(2^{i-2}k_{i-2}+2^{i-3}k_{i-3}+\ldots+2k_1+k_0)$$

A similar analysis for $k_{i-1} = 1$ shows the two submatrices in the second half of $F_{\gamma-i,\alpha}$.

It remains to show that the γ-bit binary expansion of r is that of the reverse of 2α. Note that

$$r = 2^{\gamma-2}k_{i-2} + 2^{\gamma-3}k_{i-3} + \ldots + 2^{\gamma-i+1}k_1 + 2^{\gamma-i}k_0$$

$$\alpha = 2^{i-2}k_0 + 2^{i-3}k_1 + \ldots + 2k_{i-3} + k_{i-2}$$

$$2\alpha = 2^{i-1}k_0 + 2^{i-2}k_1 + \ldots + 2^2 k_{i-3} + 2k_{i-2}$$

The γ-bit binary expansion of 2α is then

$$2^{\gamma-1}\cdot 0 + 2^{\gamma-2}\cdot 0 + \ldots + 2^i \cdot 0 + 2^{i-1}k_0 + \ldots + 2^{i-2}k_1 + \ldots +$$

$$2^2 k_{i-3} + 2^1 k_{i-2} + 2^0 \cdot 0$$

Reversing the bits of 2α amounts to making the following substitutions

$$2^1 k_{i-2} \rightarrow 2^{\gamma-2}k_{i-2}$$

$$2^2 k_{i-3} \rightarrow 2^{\gamma-3}k_{i-3}$$

$$\vdots$$

$$2^{i-1}k_0 \rightarrow 2^{(\gamma-1)-(i-1)}k_0 = 2^{\gamma-i}k_0$$

which yields the number

$$2^{\gamma-2}k_{i-2} + 2^{\gamma-3}k_{i-3} + \ldots + 2^{\gamma-i}k_0$$

which is indeed r.

Note that because of the way the indices are ordered, at the end we get

$$y[k_{\gamma-1}k_{\gamma-2}\ldots k_1k_0] = x_0[k_0k_1\ldots k_{\gamma-2}k_{\gamma-1}]$$

Hence

$$F = R_2\gamma\, F_0F_1F_2 \ldots F_{\gamma-2}F_{\gamma-1}$$

where $R_2\gamma$ is the permutation matrix corresponding to the permutation of the numbers

$$0, 1, \ldots, n-1 = 2^{\gamma}-1$$

defined by reversing the bits in the binary representation.

2. The hypercube topology

The decomposition of the DFT matrix into a product of sparse matrices, as shown in section **1** provides the essential tool for mapping the computation of the DFT on multiprocessor systems. In this section we examine the case of a hypercube.

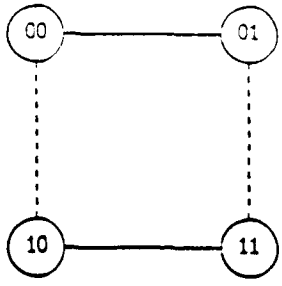
The hypercube topology has now appeared in successful commercial multiprocessor systems, including the Intel's iPSC, Amatek's S14, Ncube's *NCUBE* systems and the Connection Machines from Thinking Machines. The hypercube has a simple recursive definition:

(a) a 0-cube, with $2^0 = 1$ node, is a uniprocessor;

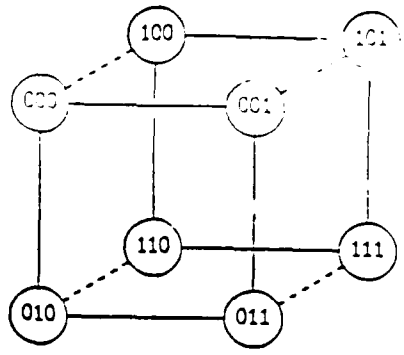
(b) for $d \geq 1$, a d-cube, with 2^d nodes, is obtained from two copies of a (d-1)-cube by connecting each pair of corresponding nodes with a communication link.

Note the very natural labeling of the nodes of the d-cube: simply precede the labels of the nodes from the first copy of the (d-1)-cube with 0, and those from the second copy with a 1 and connect nodes that differ only in the first bit.

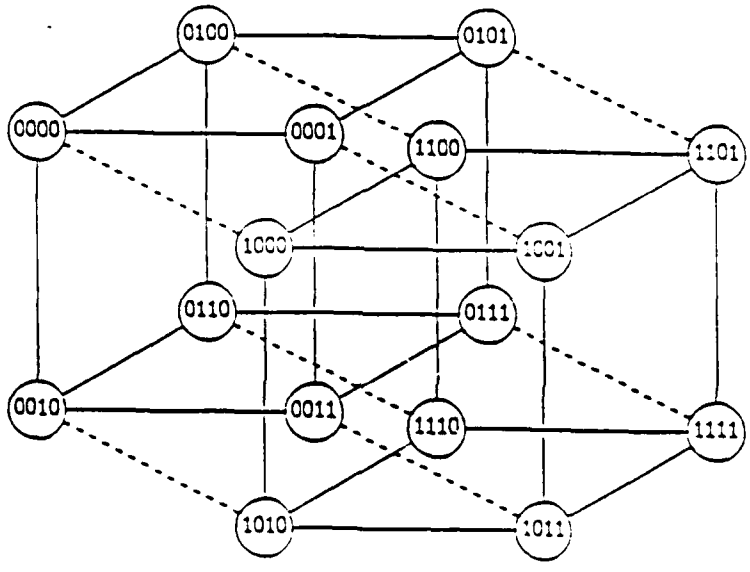
0-cube 1-cube



2-cube



3-cube



4-cube

The hypercube is attractive for several reasons, including the following

(a) many of the classical topologies (rings,trees,meshes,etc.) can be embedded in hypercubes (hence previously designed algorithms may still be used);

(b) the layout is totally symmetric (by the way, the hypercube appears among proposed architectures designed from finite algebraic groups);

(c) the communication diameter is logarithmic in the number of processors;

(d) the topology exhibits a reasonable behavior in the presence of faulty processors or communication links;

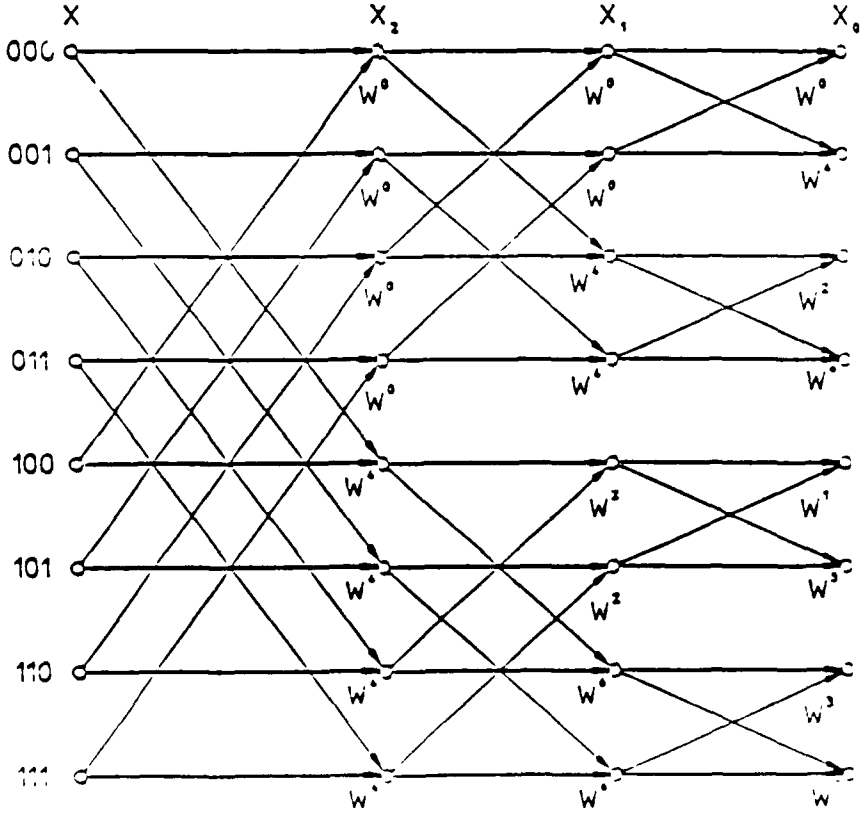
there are several paths connecting one node to another; in fact the number of disjoint paths of minimum distance from node A to node B is the Hamming distance between the (binary) labels of A and B;

(e) a hypercube is Hamiltonian, i.e a ring with 2^d nodes can be embedded in a d-cube;

(f) many algorithms for diverse fields such as the physical sciences, signal processing, numerical analysis, operations research and graphics have been successfully designed for the hypercube.

3. Example of a transform computation mapping onto a hypercube

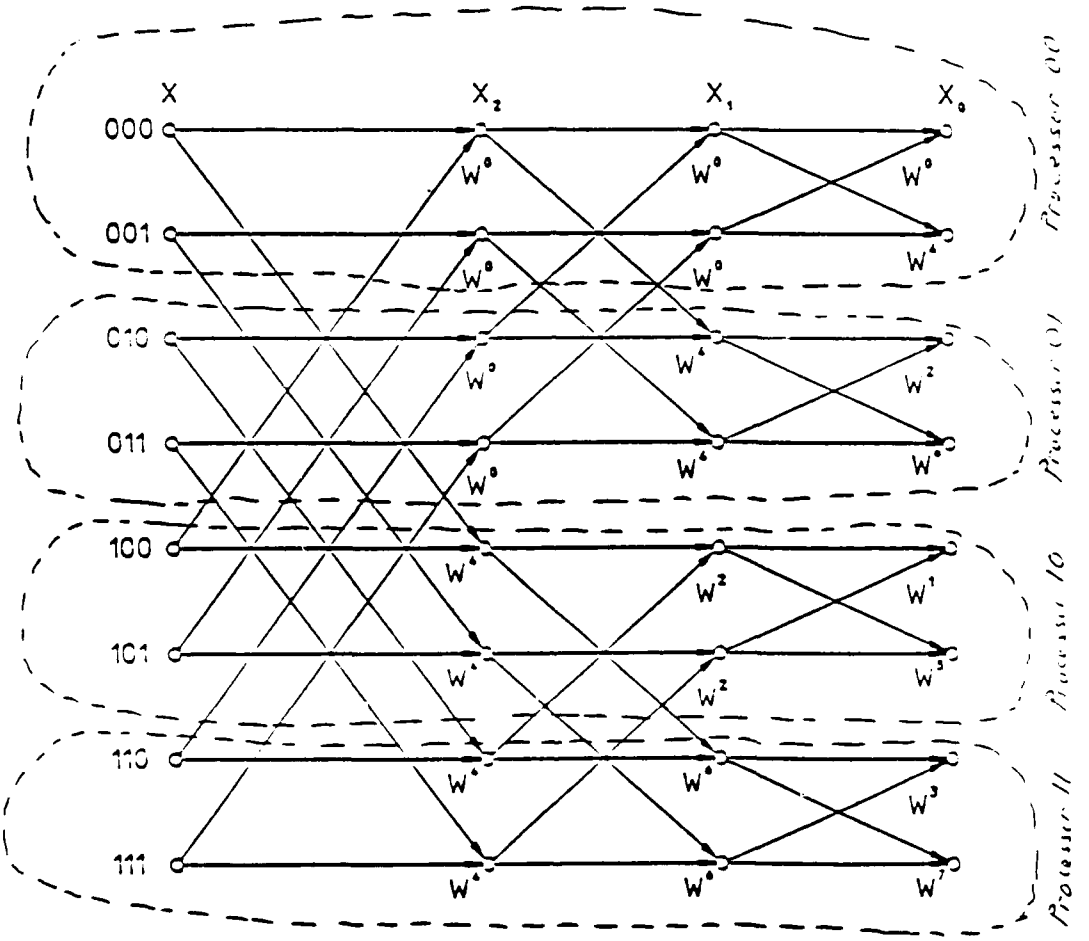
A decomposition of the DFT matrix into a product of sparse matrices is visualized, in the usual manner, by means of a signal flow graph. For the length $n = 2^3$, the decomposition of section 1 yields the following graph,



where the unscrambling of the index bits is not shown. Each column of nodes in the graph

corresponds to an intermediate vector $X_{\gamma\text{-}i}$ of section **1**.

Suppose now that a 2-cube is available. Looking at the signal flow graph, we can allocate the first 2 components (indexed 000 and 001) to processor 00, the next two components (indexed 010 and 011) to processor 01, etc. In general two components indexed ab0 and ab1 are allocated to processor ab, as shown in the following figure.



We see that the computation requires 3 steps (if we ignore the bit reversing); during the first 2 steps, interprocessor communication is required; during the last step, only local data is used in each processor, so there is no message passing between the processors.

4. The general case

The above example generalizes to the case of a length $n = 2^{\gamma}$ transform, for which a k-cube is available. Each processor is assigned a k-bit binary label. The result of section 1 allows us to allocate components of X and any intermediate vector $X_{\gamma-i}$ as follows: each processor will have $2^{\gamma}/2^{k} = 2^{\gamma-k}$ components at any stage; the processor labeled p is allocated the components with indices

$$p.2^{\gamma-k},\ p.2^{\gamma-k} + 1,\ \ldots,\ p.2^{\gamma-k} + 2^{\gamma-k}-1$$

i.e if the processor label is

$$p = p_{k-1}p_{k-2}\cdots p_1p_0$$

then the processor works on the components whose indices are

$p_{k-1}p_{k-2}\cdots p_1p_0000\ldots.00$

$p_{k-1}p_{k-2}\cdots p_1p_0000\ldots.01$

$p_{k-1}p_{k-2}\cdots p_1p_0000\ldots.10$

$p_{k-1}p_{k-2}\cdots p_1p_0000\ldots.11$

.

.

$p_{k-1}p_{k-2}\cdots p_1p_0111\ldots.10$

$p_{k-1}p_{k-2}\cdots p_1p_0111\ldots.11$

From the discussion in section 1 it should be clear that the computation can be decomposed into γ stages. The first k stages require interprocessor communication and the last γ - k stages use local data only. During stage i, where $1 \leq i \leq k$, each processor labeled

$$p = p_{k-1}p_{k-2}\cdots p_1p_0$$

communicates with and only with the processor whose label differs from p only in the ith bit from the left.

5. Decomposing with the Chinese Remainder Theorem

The scheme presented in the previous sections works for a length that is a power of 2, and should generalize easily to any power of a prime integer. The next most obvious question is how to decompose the computation if the length is not the power of a prime. In this case, a possible answer is provided by the prime factorization of n

$$n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_{k-1}^{\alpha_{k-1}} p_k^{\alpha_k}$$

and the ring isomorphism provided by the so-called Chinese Remainder Theorem (CRT).

The CRT states that if the integers $n_1, n_2, \ldots, n_k$ are pairwise coprime, the ring Z_n and the product ring

$$Z_{n_1} \times Z_{n_2} \times \ldots \times Z_{n_k}$$

are isomorphic where n is the product of the n_i, and the arithmetic in the rings is ordinary modular arithmetic (in the product everything is computed componentwise).

The forward isomorphism

$$Z_n \rightarrow Z_{n_1} \times Z_{n_2} \times \ldots \times Z_{n_k}$$

is given by

$$x \rightarrow (x \bmod n_1, x \bmod n_2, \ldots, x \bmod n_k)$$

The inverse mapping is given by

$$(a_1, a_2, \ldots, a_k) \rightarrow x$$

where

$$x = [a_1 u_1 (u_1^{-1} \bmod n_1) + a_2 u_2 (u_2^{-1} \bmod n_2) + \ldots + a_k u_k (u_k^{-1} \bmod n_k)] \bmod n$$

where $u_i = n/n_i$ and u_i^{-1} is the inverse of u_i modulo n_i (which must exist since u_i and n_i are clearly coprime).

The essence of the isomorphism is that if $n = n_1 n_2 \ldots n_k$, where the n_i are pairwise coprime, then an integer between 0 and n-1 may be thought of as a k-tuple of integers, the first between 0 and n_1-1, the second between 0 and n_2-1, ... , the last between 0 and n_k-1.

Now the relationship

$$y[a] = \sum_{b=0}^{n-1} x[a]\, W^{ab}$$

may be written

$$y(a_1,a_2,\cdots,a_k) = \sum_{b_1=0}^{n_1-1}\sum_{b_2=0}^{n_2-1}\cdots\sum_{b_k=0}^{n_k-1} x(b_1,b_2,...,b_k)\, W^{(a_1,a_2,\ldots,a_k)(b_1,b_2,\ldots,b_k)}$$

The product

$$(a_1,a_2,\cdots,a_k)(b_1,b_2,\cdots,b_k)$$

is computed in the product ring

$$Z_{n_1} \times Z_{n_2} \times \ldots \times Z_{n_k}$$

as follows

$$(a_1,a_2,\cdots,a_k)(b_1,b_2,\cdots,b_k) = (a_1b_1,a_2b_2,\cdots,a_kb_k)$$

$$= a_1b_1(1,0,0,...,0)+a_2b_2(0,1,0,...,0)+\cdots+a_kb_k(0,0,0,...,1)$$

hence if we let

$$W_i = W^{(0,0,\ldots,0,1,0,\ldots,0)}$$

where the 1 is at the ith position, then $y(a_1,a_2,\cdots,a_k)$

$$= \sum_{b_1=0}^{n_1-1}\sum_{b_2=0}^{n_2-1}\cdots\sum_{b_k=0}^{n_k-1} x(b_1,b_2,...,b_k)\, W_1^{a_1b_1} W_2^{a_2b_2}\cdots W_k^{a_kb_k}$$

$$= \sum_{b_1=0}^{n_1-1} W_1^{a_1b_1}\Big(\sum_{b_2=0}^{n_2-1} W_2^{a_2b_2}\Big(\cdots\Big(\sum_{b_k=0}^{n_k-1} x(b_1,b_2,...,b_k)\, W_k^{a_kb_k}\Big)\cdots\Big)\Big)$$

Note that W_i is an n_ith root of unity.

The computation of vector Y may thus be accomplished in k stages.

STAGE 1

$$U^1_{(b_1,b_2,\cdots,b_{k-1},a_k)} = \sum_{b_k=0}^{n_k-1} x(b_1,b_2,\cdots,b_k)\, W_k^{a_kb_k}$$

STAGE 2

$$U^2_{(b_1,\cdots,b_{k-2},a_{k-1},a_k)} = \sum_{b_{k-1}=0}^{n_{k-1}-1} U^1_{(b_1,b_2,\cdots,b_{k-1},a_k)} W_{k-1}^{a_{k-1}b_{k-1}}$$

.....

STAGE i

$$U^i_{(b_1,\cdots,b_{k-i},a_{k-i+1},a_{k-i+2},\ldots,a_k)} = \sum_{b_{k-i+1}=0}^{n_{k-i+1}-1} U^{i-1}_{(b_1,b_2,\cdots,b_{k-i+1},a_{k-i+2},\cdots,a_k)} W_{k-i+1}^{a_{k-i+1}b_{k-i+1}}$$

.....

STAGE k

$$U^k_{(a_1,a_2,\cdots,a_{k-1},a_k)} = \sum_{b_1=0}^{n_1-1} U^{k-1}_{(b_1,a_2,\cdots,a_k)} W_1^{a_1 b_1}$$

Stage i requires the computation of n/n_{k-i+1} DFTs each of length k-i+1. These DFTs may be computed in parallel, once stage i-1 has been completely finished. Note that if M_i (resp. A_i) is the number of multiplications (resp. additions) required for the n_i-transform, then in total the n-transform requires $n \cdot \sum_{i=1}^{k} \frac{M_i}{n_i}$ multiplications and $n \cdot \sum_{i=1}^{k} \frac{A_i}{n_i}$ additions.

6. Future work

We need to find decompositions of the DFT matrix that allow better mappings of the computation onto the hypercube. The scheme discussed in sections 1 through 4 for example leave the load between the processors unbalanced (some of the processors have to perform multiplications while others do not).

The decomposition of section 5 suggests a combination of pipelining and multiprocessing

that needs to be formalized. It also calls for the (parallel) study of transforms of short lengths.

7. References

[1] Fox, G. et al, Solving problems on concurrent processors, vol. I, Prentice-Hall, 1988.

[2] Kallstrom, M. and S.S. Thakkar, Programming three parallel computers, IEEE Software, Jan. 1988, pp. 11-22.

[3] Oppenheim, A.V. and R.W. Schafer, Discrete-time signal processing, Prentice-Hall, 1989.

[4] Saad,Youcef, Gaussian elimination on hypercubes, in Parallel Algorithms and Architectures (M.Cosnard et al, eds.), Elsevier Sci. Publ.,1986, pp. 5-17.

[5] Saad, Y. and M. Schultz, Topological properties of hypercubes, IEEE Trans. Computers, vol. 37, no. 7, Jul. 1988, pp. 867-872.

Conclusions

As stated at the outset there are several avenues that we are interested in pursuing further. Of these, the use of algebraic transforms and associated coding techniques in the design of neural-net based classifiers stands out as the area where we should focus research efforts next. We want to come up with classifiers capable of real time response and having low error rates. As part of this effort we need to identify, using coding-theoretic knowledge, transforms suitable for operating environments which have different design requirements: large number of classes, large block length, high fault tolerance. After establishing that the proposed family of architectures are worthwhile pursuing, we will next study how to make them more flexible by considering their concatenation with learning architectures to allow for the generation of new classes, this being achieved without prohibitively increasing the overall response time of the composite adaptive network. The last phase of the proposed effort involves the computation of the error rates for the most promising of the resulting classifiers. Complete simulations need to be carried out on realistic classification problems.

Concurrent with this report we are submitting a proposal for a follow-up project based on these considerations.

MULTILAYER TRANSFORM-BASED NEURAL NET CLASSIFIERS

1. SUMMARY

This proposal is seeking support for the investigation of neural network classifiers based on discrete transforms whose kernels can be identified with linear and nonlinear algebraic binary codes. More precisely, the study will concentrate on the feasibility of designing and implementing nonclassical classifiers that are multilayer neural nets, where one or more layer transforms points in the pattern space into points in some appropriate "frequency" domain, and where other layers classify the latter points through attributes of the transform used. Concatenation of the proposed networks with exemplar classifiers (such as adaptive resonance theory classifiers) will also be experimented with.

2. PROJECT OBJECTIVES

The main objective of the project is to produce classifiers with low error rates and real-time response. These two attributes are viewed by researchers as the most fundamental characteristics of classifiers with regard to the application of pattern classification to real-world problems such as speech processing, radar signal processing, vision and robotics. This project will explore the possibility of achieving a low error rate by viewing patterns in "frequency" domain.

Mappings from pattern space to frequency domain that are continuous must be devised, where continuous means that patterns that are close under the pattern space

metric will have their images close under the Hamming distance. The frequency domain will then appear as a code space and pattern classification might be thought of as error correction in this space.

In the first phase of the project, the codewords will be "hard-wired" among the interconnections of a neural network. The lower layers of this net will implement the foregoing continuous mapping. They transform pattern vectors into noisy versions of the codewords corresponding to the class labels. The upper layers of the net will use coding-theoretic properties in code space to correct errors in the corrupted codewords, thus identifying the class labels. Because of the hardwiring, the training time and the operational response time should be relatively very short, the latter corresponding only to the propagation delay among the layers of the neural net. Since it is not expected that a single transform (i.e algebraic code) will accommodate all classification situations, the identification of suitable transforms and their matching with different operating environments (large number of classes, high fault-tolerance, high pattern vector length, etc.) will also be studied. The large body of knowledge about algebraic codes and decoding procedures will be put to use. It should be noted that many transforms used in signal processing do arise from appropriate algebraic codes; for example, the Walsh-Hadamard transform uses the celebrated Hadamard code.

The next phase of the project will investigate the possibility of introducing more flexibility into the above architectures. Clearly for this architecture the class labels must be known beforehand, the choice of codewords that identify the labels must be built into the net, and learning from examples is nonexistent. By concatenating the proposed architecture with learning architectures, a fine-tuning of the parameters of the classifier may be performed. This approach can yield a high payoff in terms of error rates, but of course the training time and operating response time might significantly

increase.

The last phase of the project will address the computation of the error rate for the proposed classifier architecture in order to evaluate its place among proposed and implemented classifiers. A difficulty that arises here is that the continuous mapping used for going from pattern space to code space may introduce additional errors (noise).

Throughout the project, simulations will be conducted. Realistic examples of classification by neural nets are well known to require a large amount of memory and interconnect computation.
It will be interesting to find out the tradeoffs that can be achieved through the proposed architecture.

3. REFERENCES

F. J. McWilliams & N. J. A. Sloane, "The Theory of Error-Correcting Codes" , North Holland, (1977).

S. Thomas Alexander, "Adaptive Signal Processing", Springer-Verlag, (1986).

David G. Stork, "Self-Organization, Pattern Recognition, and Adaptive Resonance Networks", J. of Neural Network Computing, (Summer 1989).

R. P. Lippmann, "An Introduction to Computing with Neural Nets", IEEE ASSP Magazine, (April 1987).

MISSION

OF

ROME LABORATORY

Rome Laboratory plans and executes an interdisciplinary program in research, development, test, and technology transition in support of Air Force Command, Control, Communications and Intelligence (C^3I) activities for all Air Force platforms. It also executes selected acquisition programs in several areas of expertise. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. In addition, Rome Laboratory's technology supports other AFSC Product Divisions, the Air Force user community, and other DOD and non-DOD agencies. Rome Laboratory maintains technical competence and research programs in areas including, but not limited to, communications, command and control, battle management, intelligence information processing, computational sciences and software producibility, wide area surveillance/sensors, signal processing, solid state sciences, photonics, electromagnetic technology, superconductivity, and electronic reliability/maintainability and testability.