Pattern Recognition, Vol. 22, No. 6, pp. 733–740, 1989
Printed in Great Britain.

0031-3203/89 $3.00 + .00
Pergamon Press plc
Pattern Recognition Society

PATTERN CLASSIFICATION USING A LINEAR ASSOCIATIVE MEMORY

G. EICHMANN and T. KASPARIS

Department of Electrical Engineering, City College of City University of New York, New York, NY 10031, U.S.A.

(*Received* 6 *June* 1988; *received for publication* 5 *January* 1989)

Abstract—Pattern classification is a very important image processing task. A typical pattern classification algorithm can be broken into two parts; first, the pattern features are extracted and, second, these features are compared with a stored set of reference features until a match is found. In the second part, usually one of the several clustering algorithms or similarity measures is applied. In this paper, a new application of linear associative memory (LAM) to pattern classification problems is introduced. Here, the clustering algorithms or similarity measures are replaced by a LAM matrix multiplication. With a LAM, the reference features need not be separately stored. Since the second part of most classification algorithms is similar, a LAM standardizes the many clustering algorithms and also allows for a standard digital hardware implementation. Computer simulations on regular textures using a feature extraction algorithm achieved a high percentage of successful classification. In addition, this classification is independent of topological transformations.

Pattern classification Associative memory Topological transformation
Feature extraction Hough transform Associative encoding Iterative encoding

1. INTRODUCTION

Pattern classification problems have been studied extensively and for different classes of patterns many classification algorithms have been proposed.[1] Most classification algorithms, however, have some common features. Figure 1 shows a block diagram of a typical classification algorithm. As the first algorithm step, pattern features used in the classification are extracted. This extraction step usually varies from algorithm to algorithm. To provide a best fit for the class of patterns of interest, the way the primitives are chosen and extracted will differ. The second classification algorithm step is usually common to most algorithms. Here, once the pattern features are extracted, they are then compared with a set of stored reference features. These reference features, assumed to be *a priori* known, are established during a training phase. The process of matching the unknown features to one of the reference ones is carried out by one of the several available clustering or similarity measure algorithms.[1–3] Even though the second, classification algorithm phase, is somewhat uniform, because of the large number of implementation details, this standardization is lost. Furthermore, the implementation complexity depends on the method chosen. A dynamic clustering algorithm is time-consuming and is difficult to implement. A simple similarity measure algorithm while it is less time-consuming, in some situations it does not perform well. Additionally, no matter which method is chosen, the reference features have to be stored. It is desirable to have available a standard process that would incorporate both the reference feature storage and the decision making process.

An alternate way of matching the unknown features to the reference ones, is by mapping a feature vector into an identifying code vector, If this memory mapping is possible, then classification is accomplished. The required mapping matrix can be trained so that it will map known input and output vectors. This mapping requirement is best fulfilled by a linear associative memory (LAM) matrix. The LAM is a learning algorithm that is trained to map desired inputs into desired outputs. Additionally, because of the properties of LAM, this mapping is optimum in the least-square error sense, and it can tolerate some errors introduced by the feature extraction algorithm.

The organization of this paper is as follows. In Section 2, the background on the LAM is presented

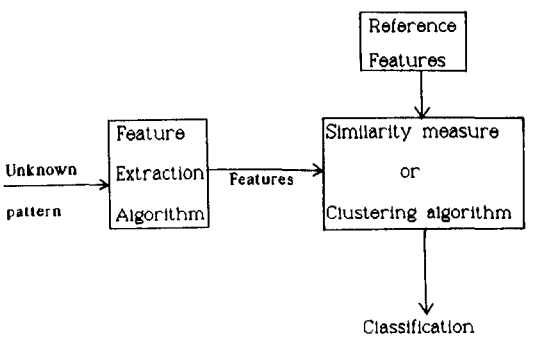

Fig. 1. Block diagram of a typical classification algorithm.

and the proposed feature encoding is described. In Section 3, a feature extraction algorithm used in our experiments is described, and the problem of texture classification is discussed. Also, a discourse on linear independence considerations is presented, and an iterative feature encoding scheme is described. Finally, in Section 4 experimental results are presented. In Section 5, the paper summary is presented.

2. BACKGROUND

Associative recall may, in general, be defined as a mapping in which a finite number of input vectors are transformed into a given set of output vectors. In the case of incomplete or erroneous input vectors, it has been shown[4] that this mapping is a least-square-error-sense optimal. It is this error tolerance that suggests its applicability to pattern (vector) restoration and classification. There are two kinds of associative recalls: the autoassociative and the heteroassociative. In the former, an incomplete vector is restored into a complete version of itself, while in the latter, in response to a given input vector an output vector is produced. The mapping between the input–output vector pair is rather arbitrary and depends on the application requirements. Associative recall suggests a working mechanism of an error-correcting content-addressable memory. This means that, for all similar vectors, in the sense of some appropriate measure, the recall will be similar to the corresponding output vector.

The mapping process is described by the following transfer relation:

$$y_k = Mx_k \qquad k = 1,\ldots,P \tag{1}$$

where x_k's and y_k's are given input and output column vectors and M is the unknown LAM matrix. The generation of a LAM is a problem of memorizing a set of responses to a set of input signals. It can also be formulated as a problem of finding an optimal matrix M solution, in the sense of least squares. Given an M, recognition is achieved by linearly transforming an unknown input, as per equation (1), and therefore it belongs to a scalar transform category. The matrix M can be determined through an iterative procedure. The mathematical details can be found in Appendix A. For the algorithm presented in Appendix A, first the X and Y matrices must be generated. This is accomplished by stacking the column feature vectors with the X columns representing the set of training features, and the Y columns the set of desired responses. In the second, recognition phase, an unknown or degraded feature vector is applied to the input of a LAM yielding an output that is similar, in a least-square-error sense, to one of the trained responses. The LAM matrix has found many applications,[4] in image correction and noise removal, in correcting for missing image fragments, etc. With proper training, the so-called novelty filter, a version of a LAM is able to detect defects in input vectors.

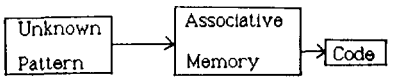

Fig. 2. An associative memory encoder.

The so-called associative encoding is a process in which a set of input patterns are mapped into a set of output codes. Figure 2 is a block diagram of an associative encoder. Comparing Fig. 2 with the more general Fig. 1, we see that, basically, both processes perform the same task. However, when an associative encoding is used for classification, then it will depend on geometrical transformations such as rotation, translation, and scaling. The classification task with a LAM is a two-stage process. In the first, a learning (or training) phase to induce the proper output responses, patterns representative of given classes are used. The LAM encoding is an elegant classifier in problems such as face recognition where geometrical transformations are not expected. However, in a more general pattern classification, this geometrical independence is very desirable. In our scheme, the LAM does not transform the pattern but encodes characteristics (features) of the pattern into an identifying code. The first advantage of this process is that we can derive the full benefit from independence properties of the feature extraction algorithm. With a properly selected feature extraction algorithm, classification can be made independent of geometrical transformations. The second advantage is that a large variety of clustering algorithms and similarity measures can be replaced by a stored vector matrix multiplication. Additionally, the reference features need not be stored in a separate memory. One drawback of this encoding process is that it requires additional development computation since after the reference features of the patterns in a class are established, the LAM has to be calculated. However, in most applications, development complexity is not critical because it is only performed once. Figure 3 shows the block diagram of a LAM-based classification algorithm. Comparing with Fig. 1 it is obvious that a LAM-based classification is more standard in the sense that the second processing block is always the same; only its content is different. An additional advantage of this standardization is that it can be implemented in hardware. The different LAM coefficients could be stored in a read-only memory (ROM). As an implementation example, next, a LAM-based texture classifier will be described.

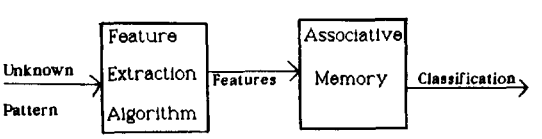

Fig. 3. A diagram of a LAM based classification algorithm.

3. A FEATURE EXTRACTION ALGORITHM

Recently, a new feature extraction algorithm suitable for regular textures has been proposed.[5] Since many regular textures consist primarily of straight lines, to detect line arrangement in the texture the algorithm uses a line detector. The Hough transform (HT)[6–8] has been used for this purpose. The HT maps straight lines into points in the HT domain and thus converts a line detection into a peak detection problem. The algorithm extracts texture features from the HT peaks and, with proper normalization, these extracted features are independent of topological transformations. The primitives used for classification are the total number of line orientations, and for each orientation the normalized average line separation. The texture feature vector $\mathbf{x}_F$ is

$$\mathbf{x}_F = [x_1, \mathbf{x}_2^t, \mathbf{x}_3^t] \tag{2}$$

where x_1 is the total line orientations, $\mathbf{x}_2$ is the normalized vector line orientations, $\mathbf{x}_3$ is the normalized vector line separations and t stands for the vector transpose. In many applications, it is desirable to minimize the feature space dimensions. In our case, by introducing some statistical data reduction analysis, the feature space can be reduced to three dimensions. It was found that if the vectors $\mathbf{x}_2$ and $\mathbf{x}_3$ are replaced by the variance of their elements, for correct classification this three-element feature vector can be used. While some classification accuracy is sacrificed, for most applications, the percentage of classification success is still high enough.

As an illustration, in Fig. 4(a) a regular line texture pattern, while in Fig. 4(b) its corresponding HT are shown. Note that the HT peak pattern follows the texture line arrangement. In Fig. 5(a) the pattern of Fig. 4(a) under topological transformations, while in Fig. 5(b) the HT of Fig. 5(a) are depicted. These figures illustrate the effects of these transformations. With proper normalization steps, the same texture features can be extracted from either Fig. 4(b) or Fig. 5(b). Figure 6(a) displays the pattern of Fig. 4(a) with uniform "salt and pepper" noise added, while in Fig. 6(b) the HT of Fig. 6(a) is shown. By comparing Fig. 6(b) with Fig. 4(b), we notice that the effect of texture noise is to raise the HT background level noise. However, it is evident from Fig. 6(b) that even in the presence of noise, successful peak detection is still possible, concluding that this feature extraction algorithm is robust. As an additional illustration, Table 1 shows the necessary normalization steps on the parameters extracted from two different views of the same texture pattern. From Table 1, it is noted that after normalization, the algorithm yields almost the same feature vector, a vector that is characteristic of this texture.

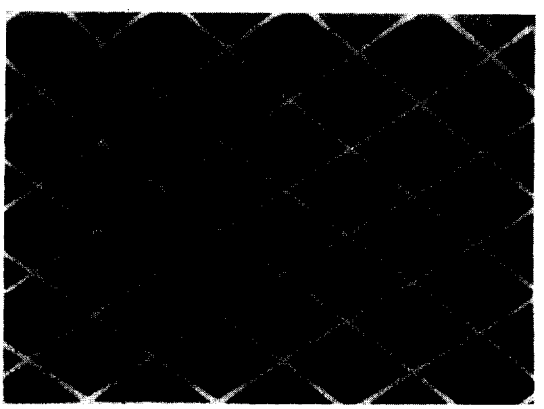

Fig. 4(a). An artificial texture pattern used as example.

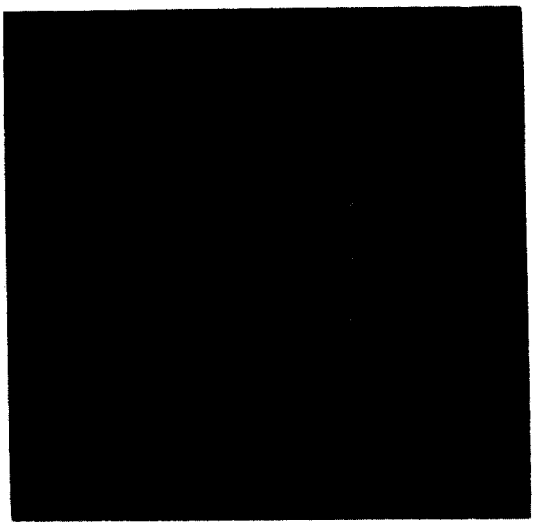

Fig. 4(b). The Hough transform of the texture of Fig. 4(a).

As a LAM-based feature classifier construction example, next, a particular texture classifier is described. For this texture classification, the input vectors are from a given class of texture reference feature vectors. One approach, for the generation of the reference features for each class, is to exercise the texture algorithm with several different texture views, where each new view is the result of a geometrical transformation on the same texture. Assuming that the classification algorithm is invariant to such transformations, the generated features are similar, although some discrepancies are to be expected. An average of all such features could be a single reference representative of that texture. However, for some texture patterns, more than one reference representative could be used. For this input, the output is assigned an orthonormal code vector, where a particular code represents the input texture, with dimensions equal to the total number of textures in a class.

An important aspect in the construction of a LAM is the linear independence of the input vectors. For LAM to be successful, the input vectors should have a high degree of linear independence. To insure linear independence, a linearly dependent input vector is not included in those iterations that generate the M matrix. Therefore, in the recall, when a degraded version of this vector is entered in the LAM, it will

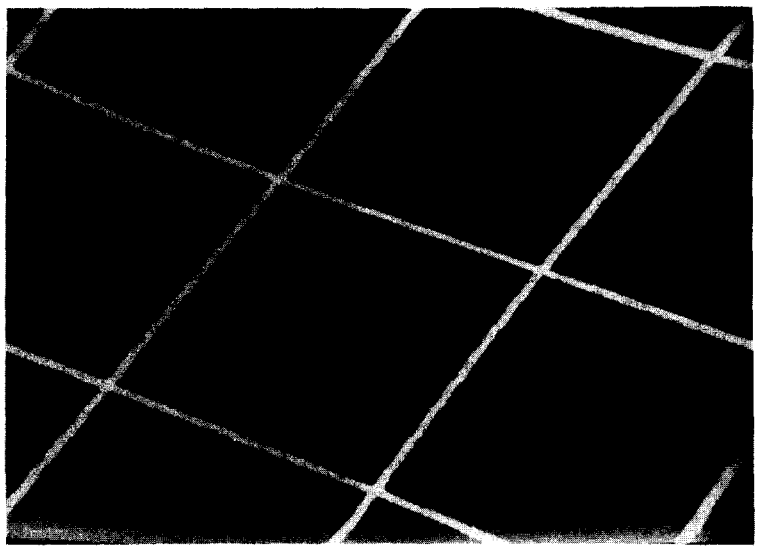

Fig. 5(a). Texture of Fig. 4(a) under a topological transformation.

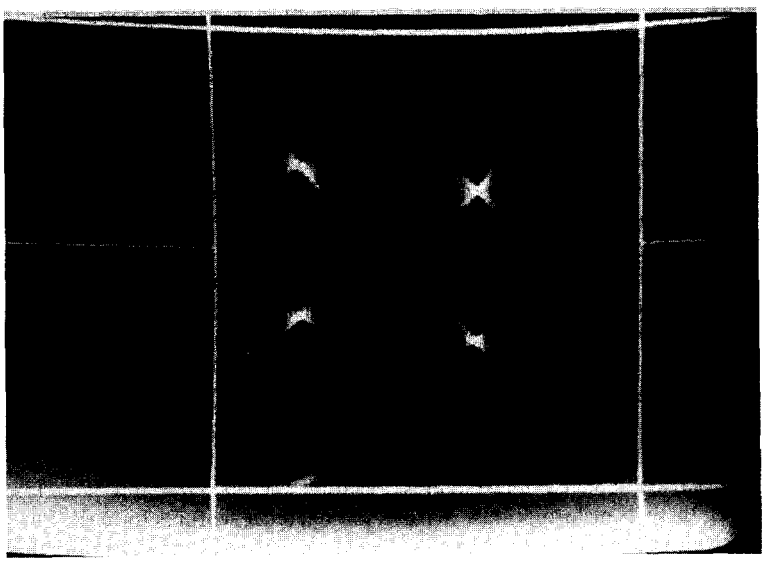

Fig. 5(b). The Hough transform of the texture of Fig. 5(a).

fail to recognize it. The result will be an incorrect output resulting in a possible misclassification.

In the texture classification algorithm, a possible output is a 3-dimensional (3D) feature vector. Since all the input feature vectors are 3D, for m texture patterns, the input X matrix is of $3 \times m$ dimensions. It is now obvious that out of the m columns of the X matrix, no more than three can be linearly independent, and therefore, only three texture patterns can correctly be classified. This represents a major drawback to this classification scheme.

There are two ways to correct this problem. First, for not too many textures in a class, we can increase the feature space dimensionality so that it will be at least m. Even though this method does not guarantee linear independence, for sufficiently different texture patterns, the resulting feature vectors will be markedly independent. However, for many different textures in a class not many independent features will be available and, thus, this is not a feasible method. A second alternative is to form an extended input matrix X'. By stacking the columns of the output below the columns of the input vectors, the extended input matrix is

$$X' = \begin{bmatrix} X \\ Y \end{bmatrix}. \tag{3}$$

The dimension of X' is $(n + m) \times m$, where n is the number of classification features and m is the total number of texture patterns in the class. Furthermore, if orthonormal code vectors are used, the Y matrix is

Fig. 6(a). The texture of Fig. 4(a) with noise added.

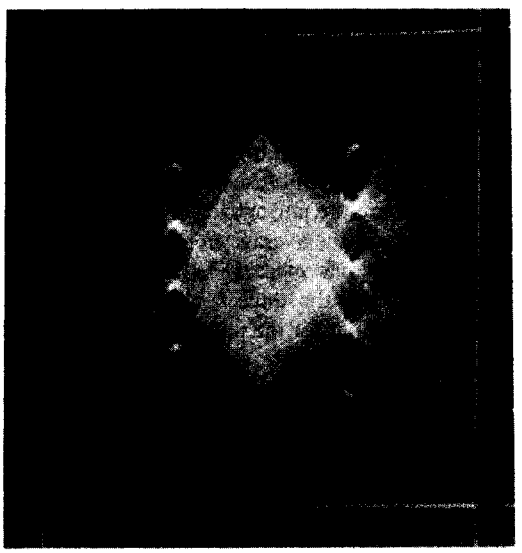

Fig. 6(b). The Hough transform of the texture of Fig. 6(a).

an identity matrix, and then all the X' columns will be linearly independent. During recall, since the code part of the input is unknown, all those elements are set to zero. This case the LAM, called to reconstruct the missing part of the input, represents an auto-associative recall.

There are two disadvantages to this scheme. First, since the input is of larger dimensions, the M matrix will also be of larger $m \times (n + m)$ as compared to $m \times n$ dimensions, leading to additional computation. Second, since now the LAM is required to correct for two sources of errors; classification algorithm errors plus an error due to a missing code element in the input vector, the classification process will be less accurate, i.e. more incorrect or unsuccessful classifications can be expected. This second disadvantage can be corrected by using an iterative LAM encoding. The iterative LAM encoding tends to improve the performance of the overall encoding system by taking

Table 1. Normalization steps applied on extracted features. The peak separations recorded at each orientation are first normalized to their maximum value, and averaged in each orientation. Then, they are circularly shifted so that the maximum value is at the leftmost position, the orientation angles are normalized, and the variance of each row is calculated

θ	52°	90°	128°	θ	72°	110°	148°
d	71	90	71	d	49	66	55
	70	0	69		51	66	54
	0	0	72		51	0	53
					0	0	53
					0	0	57

Normalize distances

52°	90°	128°	72°	110°	148°
0.79	1	0.79	0.74	1	0.83
0.78	0	0.77	0.77	1	0.82
0	0	0.80	0.77	0	0.80
			0	0	0.80
			0	0	0.86

Average distances

52°	90°	128°	72°	110°	148°
0.79	1	0.78	0.76	1	0.82

Shift circularly

90°	128°	52°	110°	148°	72°
1	0.79	0.78	1	0.82	0.76

Normalize angles

0°	38°	142°	0°	38°	142°
1	0.79	0.78	1	0.82	0.76

Normalize angles to 180°

0	0.21	0.79	0	0.21	0.79
1	0.79	0.78	1	0.82	0.76

Calculate variances	Form feature vector
$x = \{3, 0.12, 0.41\}$	$x = \{3, 0.12, 0.41\}$

into account the deterministic error of the missing code. In this method, during each new iteration, a "better" mapping M matrix is constructed.

In the first iteration step, based on the extended input X' and output Y where $Y = I$ is an identity matrix (for orthonormal codes), the matrix M_0 is calculated. The dimension of M_0 is $m \times (n + m)$. Assuming that there are no input errors, the inputs X_0' will be of the form

$$X_0' = \begin{bmatrix} X \\ 0 \end{bmatrix}. \tag{4}$$

The recall will be of the form

$$Y_0 = M_0 X_0 \tag{5}$$

where Y_0 is expected to be a close approximation of the desired output Y. We now use the output Y_0 as a new training input, i.e. $X_1 = Y_0$, and with Y as training output (the target output) form a new associative memory M_1, so that

$$Y = M_1 Y_0. \tag{6}$$

Substituting equation (5) into (6) yields

$$Y = M_1 M_0 X_0. \tag{7}$$

The new LAM matrix $M = M_1 M_0$ is now a "better" mapping matrix in that it will map the incomplete input X_0 closer to the target output Y. We also note that since the dimension of matrix M_1 is $m \times m$ so that the dimension of the matrix M is still $m \times (n + m)$. Also, since now the training input and output vectors are close to each other, the matrix M_1 will be close to the identity matrix.

The above procedure can be repeated. Using the new matrix M we can find a new output Y_1, i.e.

$$Y_1 = M_1 M_0 X_0 \tag{8}$$

where now we expect that output Y_1 will be an even closer approximation of the desired output Y. Using now Y_1 as a training input and Y (the desired output) as the training output, a new M_2 matrix can be generated. Since now Y_1 is even closer to Y, then we expect that M_2 will be even closer to the identity matrix, and the new, improved, mapping matrix as $M = M_2 M_1 M_0$.

If this procedure is repeated several times, then the training input and output will be so close to each other that their associative memory matrix will converge to the identity matrix. After k iterations the overall mapping matrix will be

$$M = M_k M_{k-1} \ldots M_1 M_0 \tag{9}$$

where M matrices with higher index are closer to the identity matrix.

Two possible variations of this scheme are the following. To take algorithm imperfections into account, at the start of the iterations, in equation (4), noise is added to matrix X. In a second variation, instead of using an approximate output as a training input, in the next iteration, the combination of feature matrix-approximate code, i.e. the training input during the kth iteration

$$X_k' = \begin{bmatrix} X \\ Y_{k-1} \end{bmatrix} \tag{10}$$

instead of Y_{k-1} is used. In this case the combined M matrix could be the result of the kth iteration step. One basic problem with this iterative LAM scheme is still the issue of linear independence. If at any iteration, one training input vector is linearly dependent on the others, then that column of the M matrix will not converge to the correct one, and the corresponding output code will be incorrect.

4. EXPERIMENTAL RESULTS

Two types of experiments were performed. In the first type, using a nine-element feature vector, eight different textures were described. In the second type, using a three element feature vector, the same eight texture patterns were described. Since the test texture patterns add up to a maximum of four line directions, a nine-element feature vector, where the first element represents the number of detected directions, the next four elements are the normalized angles, and the last four elements are the average normalized separation in each direction, is formed. If less than four directions are detected, then the unused entries are filled by zeroes. For example, for the pattern depicted in Table 1, the nine-element feature vector is:

$$\mathbf{x}_F = [3, 0, 0.21, 0.79, 0, 1, 0.79, 0.78, 0].$$

An eight element set of orthonormal vectors are used as code vectors. By using the algorithm four times on four different versions of the same texture pattern and averaging the resulting feature vectors, the reference feature vectors were generated. Using these reference feature vectors as the training inputs and their corresponding code vectors as the training outputs, a LAM matrix is calculated. The performance of this LAM memory was tested by either submitting an unknown feature vector generated by the texture classification algorithm, or by simply adding noise to one of the training inputs and using it as a test input. In both cases, the test input is submitted to the LAM and the desired information is obtained by locating the position of the maximum element in the output. In the event of two equal peaks at different locations of the output vector, then a "don't know" is declared.

Extensive testing of this scheme found the LAM to be very successful, with an average of 95% correct classification. Most of the unsuccessful attempts were "don't know's", although some incorrect cases were also observed. As an illustration, the reference feature vector (training input) of one of the patterns used in our experiments was:

$$\mathbf{x}_F = [3, 0, 0.20, 0.80, 0, 1, 0.77, 0.79, 0].$$

The testing input for this pattern was:

$$\mathbf{x} = [3, 0, 0.26, 0.84, 0, 1, 0.70, 0.83, 0].$$

The code vector (training output) of this pattern was:

$$\mathbf{y} = [0, 0, 0, 1, 0, 0, 0, 0].$$

The resulting LAM output was

$$\mathbf{y}' = [0.4, -0.2, 0.3, 0.8, 0.1, -0.1, 0.6, 0.2].$$

By comparing the LAM output with the code vector we see that this is a correct classification case.

For the second type of experiments, to describe the patterns, only three numbers were used (see Table 1). The training input was formed by stacking the three classification numbers together with the code vector into a single vector, while the training output was again the code vector. For testing, the unknown feature vector was concatenated to a zero vector and submitted to the LAM. The desired information was obtained by finding the position of the maximum element in the output vector. Again, extensive testing found this scheme to be less successful than the previous one; however, its performance can be considered satisfactory. The average percentage of classification success rate was estimated to 70%. By using iterative encoding, this percentage can be increased to about 80%. It should be noted that the reported classification success rate of other, far more elaborate algorithms, averages between 46% to an 82% maximum with a median value of around 60%.[9] In addition, compared to elaborate clustering algorithms, this method is simpler to implement. One additional comment is that in this type of experiments about half of the failures were due to incorrect classification while the other half were "don't know" types. Again, for the purpose of illustration, the training input of a pattern was:

$$\mathbf{x} = [3, 0.12, 0.41, 0, 0, 0, 1, 0, 0, 0, 0]$$

where the testing input was:

$$\mathbf{x}' = [3, 0.11, 0.42, 0, 0, 0, 0, 0, 0, 0, 0].$$

The texture code vector was:

$$\mathbf{y} = [0, 0, 0, 1, 0, 0, 0, 0].$$

The LAM output was:

$$\mathbf{y}' = [0.2, 0.3, 0.1, 0.7, -0.2, 0.5, 0.4, 0.1].$$

When iterative encoding was used, the LAM output was:

$$\mathbf{y}' = [0.1, 0.2, -0.1, 0.8, 0.1, -0.1, 0.1, -0.2].$$

Notice that this is also a successful classification case.

5. SUMMARY

A new application of a LAM was introduced. In this application, the LAM is used as a learning encoder to encode pattern features into identifying code tag vectors. As compared to associative encoding, LAM is teamed with a feature extraction algorithm to produce a classification system. As opposed to previous methods, LAM replaces tedious clustering algorithms and similarity measures with a stored vector matrix multiplication. Additionally, the reference features are part of the calculated memory, and need not be stored separately. This scheme standardizes classification algorithms and decreases computation time, since it can be implemented in fast hardware as well. Extensive simulation showed a high degree of accuracy which can be further enhanced by using a new iterative encoding scheme.

Acknowledgement—The grant support of the U.S. Air Force Office of Scientific Research is gratefully acknowledged.

REFERENCES

1. T. Y. Young and T. W. Calvert, *Classification, Estimation and Pattern Recognition*. Elsevier, New York (1974).
2. E. Diday and J. Simon, Clustering analysis, *Digital Pattern Recognition*, K. S. Fu, Ed., Springer, New York (1980).
3. B. Everitt, *Cluster Analysis*. Halsted Press, New York (1980).
4. T. Kohonen, *Self Organization and Associative Memory*. Springer, New York (1984).
5. G. Eichmann and T. Kasparis, Topologically invariant texture descriptors, *Comput. Vision Graphics Image Process.* **41**, 267–282 (1988).
6. P. V. C. Hough, Method and Means for Recognizing Complex Patterns, U.S. Patent 3,069,654 (1962).
7. R. O. Duda and P. E. Hart, Use of the Hough Transformation to detect lines and curves in pictures, *Communs Ass. Comput. Mach.* **15**, 11–15 (1972).
8. J. Sklansky, On the Hough technique for curve detection, *IEEE Trans. Comput.* **C-24**, 923–926 (1978).
9. B. P. Kjell and C. R. Dyer, Edge separation and orientation texture measures, *IEEE Conf. Rec. Pattern Recognition*, pp. 306–311 (1985).
10. T. N. E. Greville, Some applications of the pseudo-inverse of a matrix, *SIAM Rev.* **2**, 15–22 (1960).
11. A. Albert, *Regression and the Moore-Penrose Pseudo Inverse*. Academic Press, New York (1972).

APPENDIX 1

The associative memory mapping of an input vector x_k into an output vector y_k can be described by the transfer relation:

$$y_k = Mx_k \qquad k = 1, \ldots, p \tag{A1}$$

where M is the unknown LAM. An iterative method for the determination of the matrix M will now be presented.

Let (x_k, y_k), $k = 1, \ldots, P$, be two sets of vectors. Forming the two rectangular matrices X and Y, with the vectors x_k and y_k as their columns, as

$$X = [x_1, x_2, \ldots, x_p], \quad Y = [y_1, y_2, \ldots, y_p], \tag{A2}$$

every pair of vectors from sets (x_k, y_k) can be related by the matrix equation

$$Y = MX. \tag{A3}$$

Using the so-called "Moore–Penrose" method,[10] the least squares optimal solution of equation (A3) is found as

$$M = YX^{+} \tag{A4}$$

where X^+ is the pseudo-inverse of X. The iterative solution of equation (A4) has the form of a difference equation[4]

$$M_k = M_{k-1} + (y_k - M_{k-1}x_k)[C_k]^t \quad \text{(A5)}$$

where M_k and M_{k-1} are the new and previous optimal matrices, x_k and y_k are the new observation vectors, where t denotes the transpose and $[C_k]^t$ is a gain vector that defines the correction. The iterations may be initiated with either $M_0 = 0$ or $M_0 = I$ where I is the identity matrix. The expression for the gain factor is:

$$[C_k]^t = [h_k]^t/\|h_k\|^2 \text{ for } \|h_k\| \neq 0 \quad \text{(A6)}$$

where h_k is the orthonormal projection of x_k on the subspace spanned by the vectors $h_1, \ldots, h_{k-1}$. The orthogonal projection of the vector x_k to the set of previous columns of the matrix $H = [h_1, h_2, \ldots, h_p]$ can be obtained by using the Gram–Schmidt orthogonalization[11]

$$h_k = x_k - \sum_{i=1}^{k-1} (x_k, h_i) \cdot h_i/\|h_i\|^2, \quad \|h_i\| \neq 0 \quad \text{(A7)}$$

where (x_k, h_i) is the vector inner product, and $\|h_i\|$ is the Euclidean norm of the vector h_i.

About the Author—GEORGE EICHMANN received the B.E.E. and M.E.E. degrees from The City College of New York and the Ph.D. (E.E.) from The City University of New York, respectively. He joined the Department of Electrical Engineering at the City College of New York in 1968 as an Assistant Professor. At City College he held positions of Associate Professor and full Professor. He also served as Department Chair from 1982 to 1988. Presently he holds the title of Herbert Kayser Professor of Electrical Engineering. His research interest is in pattern analysis computer vision, and optical computing. Dr Eichmann is a member of the IEEE, OSA, and SPIE.

About the Author—TAKIS KASPARIS was born in Cyprus in 1956. He received the Diploma in Electrical Engineering from the National Technical University of Athens in 1980, and the M.E. and Ph.D. degrees from The City College of The City University of New York in 1982 and 1988 respectively, also in Electrical Engineering. While at City College, he held the positions of Graduate Fellow, Adjunct Lecturer, and Instructor. He is currently an Electronics Design Engineer and consultant in the fields of Image Processing and Electronics. His research interests include non-linear filtering algorithms, pattern analysis and computer vision. Dr Kasparis is a member of the Technical Chamber of Greece.