International Journal of Neural Systems, Vol. 2, No. 3 (1991) 169–184

NEURAL NETS FOR DUAL SUBSPACE PATTERN RECOGNITION METHOD

Lei Xu* and Adam Krzyzak
Centre for Pattern Recognition & Machine Intelligence, Dept. of Computer Science, Concordia University, 1455 deMaisonneuve Blvd. West, Montreal, PQ, Canada H3G 1M8

Erkki Oja
Lappeenranta University of Technology, Department of Information Technology Box 20, 53851 Lappeenranta, Finland

Received 26 July 1991

A new modification of the subspace pattern recognition method, called the dual subspace pattern recognition (DSPR) method, is proposed, and neural network models combining both constrained Hebbian and anti-Hebbian learning rules are developed for implementing the DSPR method. An experimental comparison is made by using our model and a three-layer forward net with backpropagation learning. The results illustrate that our model can outperform the backpropagation model in suitable applications.

1. Introduction

Principal Component Analysis (PCA) is a powerful data compression and data analysis tool in multivariate statistics literature. It has also been widely studied and used in signal processing and pattern recognition literature. Oja[1] showed in 1982 that a simple linear unit with a constrained Hebbian learning rule can extract the first principal component from a set of stationary input data. Since the recent renaissance of neural network research, there appears to be an increasing interest in investigating the connection between neural networks and PCA. Several new developments have been made. For example: (1) Neural network models have been proposed for extracting several principal components[2–5] instead of the first principal component in the earlier work. (2) It has been shown that by minimizing a quadratic error energy, a linear, three-layer feedforward net performs PCA.[6] (3) Neural nets or PCA have been further extended to perform the so-called constrained PCA which can avoid the undesirable redundant components and noisy components.[7] A good review about PCA nets was recently given by Baldi and Hornik,[8] and some convergence analysis was done by Hornik and Kuan.[9]

PCA also played a key role in the earlier version of subspace pattern recognition method (SPRM)[10] in which each pattern class is represented by a subspace spanned by a group of basis vectors—the orthogonal components obtained by PCA. SPRM was later developed further by a number of authors. One important development was that nonorthogonal vector bases are used as the base vectors of subspaces in a method called the Learning Subspace Method (LSM)[11] and its modified version ALSM.[12] Both PCA and SPRM were brought into the connections with neural network literature by Oja and Kohonen.[1,13,14]

In both PCA and SPRM, the principal components (i.e., directions on which the data have large variances) are regarded as important while those components which have small variances, for convenience we call them minor components, are often regarded as unimportant or as noise. However, in some cases the minor components are of the same importance as principal components. For example, they have been used in signal processing literature, especially in the Pisarenko method of spectral estimation.[15] Recently, we have shown that the minor components can play an important role in a classical statistical problem: curve or hypersurface fitting.[16] The problem is often encountered in many engineering problems as well as cognitive perception problems (e.g., computer vision). By using a neural unit with a constrained anti-Hebbian learning, this fitting problem can be solved with significantly improved performance compared to that obtained by the usual least square method. In this paper, we will further show that pattern classes could

* On leave from Dept. of Mathematics, Peking University, Beijing, P.R. China. During Feb. 1989–May 1990, he was with Lappeeranta University of Technology. Presently he is with Harvard University, Division of Applied Sciences.

be well represented not only by its subspace with its basis vectors being principal components, but also by its complementary subspace with its basis vectors being minor components. It is better to represent pattern classes by both types of subspaces so that much computing time and storage could be saved. We call such a representation dual subspace and its pattern recognition as Dual Subspace Pattern Recognition (DSPR). The nice thing is that all the techniques (including LSM) of the conventional subspace method can be adapted with minor changes to DSPR. Furthermore, we propose two neural-net models for implementing DSPR. Finally we demonstrate through computer simulations that for some applications one can obtain better results by using our model than by using the popular backpropagation approach.

2. Dual Subspace Pattern Representation and Classification Rule

In pattern recognition, the primary goal is to build an explicit or implicit model for representing the pattern classes such that classification error for new data is minimized, or equivalently, the generalization ability of the classifier is maximized. In the subspace methods of pattern recognition,[10–12,17] the primary model for class representation is a subspace in the n-dimensional Euclidean vector space R^n with each pattern object represented as a vector of n real-valued elements. The classification criterion for a pattern $\mathbf{x}$ is its orthogonal distance from the class subspace, i.e., $\mathbf{x}$ is classified into a class which gives the shortest such distance.

Any set of p linearly independent vectors $\mathbf{u}_1, \ldots, \mathbf{u}_p$ in R^n (with $p < n$) spans a subspace, say L, which is the set

$$L = L(\mathbf{u}_1, \ldots, \mathbf{u}_p) = \left\{ \mathbf{x} \,\middle|\, \mathbf{x} = \sum_{i=1}^{p} \alpha_i \mathbf{u}_i, \text{ for some scalars } \alpha_1, \ldots, \alpha_p \right\}. \tag{1}$$

Practically, the common way for storing L in a computer is to store its basis vectors $\mathbf{u}_1, \ldots, \mathbf{u}_p$.

The distance of $\mathbf{x}$ from L is defined by

$$d(\mathbf{x}, L) = \|\bar{\mathbf{x}}\|^2 / \|\mathbf{x}\|^2 , \tag{2a}$$

$$\bar{\mathbf{x}} = (I - P)\mathbf{x}, \quad \mathbf{x} = \bar{\mathbf{x}} + \hat{\mathbf{x}}, \quad \hat{\mathbf{x}} = P\mathbf{x}, \quad \hat{\mathbf{x}} \perp \bar{\mathbf{x}}$$

$$P = A(A^tA)^{-1}A^t, \qquad A = [\mathbf{u}_1 \ldots \mathbf{u}_p] \tag{2b}$$

i.e., $\hat{\mathbf{x}}$ is the projection of $\mathbf{x}$ on L, and $\bar{\mathbf{x}}$ is the corresponding orthogonal residual component, and P is the project matrix of L.

Specifically, when, $\mathbf{u}_1, \ldots, \mathbf{u}_p$ are orthonormal, since $P = \sum_{i=1}^{p} \mathbf{u}_i\mathbf{u}_i^t$, we have

$$\bar{\mathbf{x}} = \mathbf{x} - \hat{\mathbf{x}} = \mathbf{x} - \sum_{i=1}^{p} (\mathbf{x}^t\mathbf{u}_i)\mathbf{u}_i . \tag{2c}$$

In this case, the computation could be simplified considerably. For this reason, when $\mathbf{u}_1, \ldots, \mathbf{u}_p$ are nonorthonormal, they are usually normalized by the Gram–Schmidt orthonormalization method.

Assume that there are K pattern classes $\omega_1, \ldots, \omega_K$, each of which is represented by its own class subspace and basis vectors

$$L_k = L(\mathbf{u}_1^{(k)}, \ldots, \mathbf{u}_{p_k}^{(k)}), \qquad k = 1, \ldots, K$$

or equivalently by the corresponding projection matrices $P_1, \ldots, P_k$. From Eq. (2a), the classification rule for a new pattern $\mathbf{x}$ is then

if $d(\mathbf{x}, L_i) < d(\mathbf{x}, L_j)$, for all $j \neq i$,

then classify $\mathbf{x}$ to ω_i . (3a)

Furthermore, from Eqs. (2b) and (2c), the rule could be simplified into the following form:

if $\delta_i(\mathbf{x}) > \delta_j(\mathbf{x})$, for all $j \neq i$,

then classify $\mathbf{x}$ to ω_i . (3b)

$$\delta_j(\mathbf{x}) = \begin{cases} \mathbf{x}^tP_j\mathbf{x}, & \text{when } \mathbf{u}_1^{(j)}, \ldots, \mathbf{u}_{p_j}^{(j)} \text{ are nonorthonormal;} \\ \sum_{i=1}^{p_j} (\mathbf{x}^tu_i^{(j)})^2 & \text{when } \mathbf{u}_1^{(j)}, \ldots, \mathbf{u}_{p_j}^{(j)} \text{ are orthonormal.} \end{cases} \tag{3c}$$

So we see that a subspace classifier is quite simple and the key problem for training such a classifier is how to get the basis vectors $\mathbf{u}_1^{(j)}, \ldots, \mathbf{u}_K^{(j)}$ for each class ω_j.

The above describes the conventional form of subspace pattern recognition method. In the following, we further extend this form into a more general form which is based on what we call the dual subspace representations. In this new form, on one hand, some classes are represented by the subspaces which are the same as those described above, i.e., a class is repre-

sented by the basis vectors which span the same subspace as that spanned by the sample pattern vectors of the class. For clarity, we call such a subspace the *own subspace of the class*, or in short the *own subspace*. On the other hand, some classes are represented by the orthogonal complementary subspaces of their own subspace of the corresponding class (i.e., a class is represented by the basis vectors which are orthogonal to the sample pattern vectors of the class). We call such a subspace the *complementary subspace of the class*, or in short the *complementary subspace*.

Let us observe an example. We consider a simple hyperplane:

$$a_1\mathbf{x}_1 + a_2\mathbf{x}_2 + \cdots + a_n\mathbf{x}_n = 0 . \tag{4}$$

Suppose that there is a pattern class and all the samples of the class are located on this hyperplane; then with the conventional form of subspace representation, the dimension of its own subspace is $n - 1$, i.e., it needs at least $n - 1$ basis vectors to span the subspace for representing the class. This is obviously not an economical way for the class representation. From hyperplane fitting point of view, only one vector $\mathbf{a} = [a_1, a_2, \ldots, a_n]^t$ is enough to represent the hyperplane, and thus the pattern class located on this hyperplane. This $\mathbf{a}$, being orthogonal to all the samples of the class, spans a subspace $a = \{\mathbf{x} \mid \mathbf{x} = \alpha\mathbf{a}$, where α is a real scalar$\}$ which is just the orthogonal complementary subspace of its own subspace. From this example, we could see that it would be better to solve the basis vectors of the complementary subspace than to solve the basis vectors of the own subspace for representing some pattern classes, since it needs less storage and computations for pattern classification [see Eq. (3c)].

In fact, for any pattern class ω_i, we always have two choices for representing the class. One is to use the p_i basis vectors $\mathbf{u}_1^{(i)}, \ldots, \mathbf{u}_{p_i}^{(i)}$ of the own subspace L_i; the other is to use the $q_i = n - p_i$ basis vectors $\mathbf{v}_1^{(i)}, \ldots, \mathbf{v}_{q_i}^{(i)}$ of the complementary subspace $\bar{L}_i$, where, $\bar{L}_i \perp L_i$, $\bar{L}_i + L_i = R^n$. If we denote $\bar{P}_i = B_i(B_i^tB_i)^{(-1)}B_i^t$, $B_i = [\mathbf{v}_1^{(i)} \ldots \mathbf{v}_{q_i}^{(i)}]$, we have

$$\hat{\mathbf{x}} = P\mathbf{x} = (I - \bar{P})\mathbf{x}, \quad \mathbf{x} = \bar{P}\mathbf{x} = (I - P)\mathbf{x}, \quad \hat{\mathbf{x}} \perp \bar{\mathbf{x}}$$

$$d(\mathbf{x}, L_i) = 1 - d(\mathbf{x}, \bar{L}_i) = \|\bar{\mathbf{x}}\|^2/\|\mathbf{x}\| . \tag{5a}$$

Thus, it is not difficult to see that the following discriminant rule gives the equivalent effects as rule Eq. (3b) gives

$$\textbf{if } \bar{\delta}_i(\mathbf{x}) < \bar{\delta}_j(\mathbf{x}), \quad \text{for all } j \neq i,$$

$$\textbf{then classify } \mathbf{x} \text{ to } \omega_i . \tag{5b}$$

$$\bar{\delta}_j(\mathbf{x}) = \begin{cases} \mathbf{x}^t\bar{P}_j\mathbf{x}, & \text{when } \mathbf{v}_1^{(j)}, \ldots, \mathbf{v}_{n-p_j}^{(j)} \text{ are nonorthonormal;} \\ \sum_{i=1}^{n-p_j} (\mathbf{x}^t\mathbf{v}_i^{(j)})^2 & \text{when } \mathbf{v}_1^{(j)}, \ldots, \mathbf{v}_{n-p_j}^{(j)} \text{ are orthonormal.} \end{cases} \tag{5c}$$

where $\mathbf{v}_1^{(j)}, \ldots, \mathbf{v}_{n-p_j}^{(j)}$ are basis vectors of the complementary subspace $\bar{L}_j$ of class ω_j.

It follows from Eq. (3c) and Eq. (5c) that the own subspace L_j is a better choice for representing a class ω_j if $p_j \leq n/2$; otherwise, the complementary subspace $\bar{L}_j$ is a better choice. The advantages of using the complementary subspace for representation become more obvious when p_j goes near n, since it follows from Eqs. (3c) and (5c) that considerable storage and computations could be saved for pattern classification. To take the advantages of both choices, we divide the K pattern classes into two groups $G_1 = \{\omega_i \mid$ *with the correspondent* $p_i \leq n/2\}$ and $G_2 = \{\omega_i \mid$ *with the correspondent* $p_i > n/2\}$. Then, for all $\omega_j \in G_1$, the basis vectors $\mathbf{u}_1^{(j)}, \ldots, \mathbf{u}_{p_j}^{(j)}$ of the own subspace are sought for representing the class, and for all $\omega_j \in G_2$, the basis vectors $\mathbf{v}_1^{(j)}, \ldots, \mathbf{v}_{n-p_j}^{(j)}$ for the complementary subspace are sought for representing the class. We call this kind of hybrid subspace representation as the dual subspace representation of pattern space. Under the dual subspace representation, the discriminant rule is given by

$$\mathbf{x} \text{ is classified into } \begin{cases} \omega_k, & \text{if } \delta_k(\mathbf{x}) > \|\mathbf{x}\|^2 - \bar{\delta}_{k'}(\mathbf{x}) ; \\ \omega_{k'}, & \text{otherwise.} \end{cases} \tag{6}$$

$$\delta_k(\mathbf{x}) = \max \delta_i(\mathbf{x}), \quad \forall\omega_i \in G_1$$

and

$$\bar{\delta}_{k'} = \min \bar{\delta}_i(\mathbf{x}), \quad \forall\omega_i \in G_2 .$$

The result is derived from the fact that

$$\bar{\delta}_j(\mathbf{x}) = \mathbf{x}^t\bar{P}_j\mathbf{x} = \mathbf{x}^t(I - P_j)\mathbf{x}$$

$$= \|\mathbf{x}\|^2 - \mathbf{x}^tP_j\mathbf{x} = \|\mathbf{x}\|^2 - \delta_j(\mathbf{x}) .$$

3. Algorithms for Implementing Dual Subspace Pattern Recognition

As extensively discussed by Oja,[17] a number of algorithms have been developed for implementing the subspace methods described by Eqs. (3). Basically, they can be grouped into the following two types:

(1) The basis vectors of each class are obtained from the principal component analysis (PCA) (or equivalently K–L transform) on the training set of the class, i.e., the basis vectors of class ω_i are given by the first p principal eigenvectors of the class correlation matrix $R_i = E[\mathbf{x}\mathbf{x}^t | \mathbf{x} \in \omega_i]$ with the p predetermined heuristically or by some criterion, e.g.,

$$\gamma = \sum_{i=p+1}^{n} \lambda_i \Big/ \sum_{i=1}^{n} \lambda_i \leq \kappa \ ,$$

$$\lambda_1 \geq \cdots \geq \lambda_n \geq 0 \text{ are eigenvalues of } \sum_i$$

where $0 < \kappa < 1$ is a threshold. The representative of the algorithm of this type is CLAFIC proposed by Watanabe.[10]

(2) The Kohonen's learning subspace method (LSM)[11] and variants.[12] By this method, the initial estimations of the own subspaces L_i and its dimension p_i of each class ω_i, $i = 1, \ldots, K$ are first roughly determined by some means (e.g., by using CLAFIC); then, for each training sample $\mathbf{x}$ with a known label o (i.e., from class ω_o), find a class ω_r with

$$\delta_r(\mathbf{x}) = \max \delta_i(\mathbf{x}), \quad \forall \omega_i \in \{\omega_1, \ldots, \omega_K\} - \{\omega_o\} \ . \tag{7a}$$

Then, the subspace L_o and L_r are rotated into $L_o + \Delta L_o$ and $L_r + \Delta L_r$ by

$$\begin{aligned} \Delta L_o &= \alpha_o \mathbf{x}\mathbf{x}^t L_o \ , \\ \Delta L_r &= -\alpha_r \mathbf{x}\mathbf{x}^t L_r \ , \end{aligned} \tag{7b}$$

where α_o and α_r are both positive and $\alpha_r < (\mathbf{x}^t\mathbf{x})^{(-1)}$. They are determined to guarantee that

$$d(\mathbf{x}, L_o + \Delta L_o) < d(\mathbf{x}, L_o)$$

and

$$d(\mathbf{x}, L_r + \Delta L_r) > d(\mathbf{x}, L_r) \ .$$

Basically speaking, all the existing algorithms of both types could be adapted to the dual subspace representation with slight modifications. For algorithms of the first type, the modification is made by choosing the first p principal eigenvectors of the class correlation matrix R_i when $p \leq n/2$ and the last $n - p$ eigenvectors when $p > n/2$. For the algorithms of the second type, the modifications are to replace Eqs. (7a) and (7b) by Eqs. (8a) and (8b) given as follows:

$$\delta_r(\mathbf{x}) = \max\{\delta_k^{(r)}, \|\mathbf{x}\|^2 - \bar{\delta}_k^{(r)}(\mathbf{x})\}$$

$$\delta_k^{(r)}(\mathbf{x}) = \max \delta_i(\mathbf{x}), \ \forall \omega i \in G_1 \quad \text{and } \omega_i \neq \omega_o \tag{8a}$$

$$\bar{\delta}_k^{(r)}(\mathbf{x}) = \min \bar{\delta}_i(\mathbf{x}), \ \forall \omega i \in G_2 \quad \text{and } \omega_i \neq \omega_o$$

and

$$\begin{aligned} \Delta L_o &= \begin{cases} \alpha_o \mathbf{x}\mathbf{x}^t L_o, & \text{when } p_o < n/2 \\ -\alpha_o \mathbf{x}\mathbf{x}^t L_o, & \text{when } p_o \geq n/2 \ , \end{cases} \\ \Delta L_r &= \begin{cases} -\alpha_r \mathbf{x}\mathbf{x}^t L_r, & \text{when } p_r < n/2 \\ \alpha_r \mathbf{x}\mathbf{x}^t L_r, & \text{when } p_r \geq n/2 \ , \end{cases} \end{aligned} \tag{8b}$$

where p_o and p_r are the dimensions of L_o and L_r, respectively.

In the following, we do not intend to discuss the detailed steps of the above modifications. Instead, we will propose two neural network models for implementing the dual subspace pattern recognition.

4. Neural Nets for Dual Subspace Pattern Recognition

Oja and Kohonen suggested[13] that the subspace pattern representation and the learning algorithm are pertinent to neural networks too. Moreover, Oja recently proposed[14] a subspace network for extracting the basis vectors of the own subspace of one class, and then used K such networks as K modules for classification. In this paper, we further propose two neural network models (including some versions) for dual subspace pattern recognition (DSPR), through adaptively extracting the orthogonal basis vectors to represent a subspace. One is based on a combination of Oja's Stochastic Gradient Ascent (SGA) learning algorithm[4] and its anti-Hebbian learning counterpart (here, it is called ASGA). The other is based on a combination of Rubner's asymmetric lateral inhibition algorithm[3] (here, simply denoted by ALIA) and its anti-Hebbian learning counterpart (denoted by AHALIA).

4.1. *Model 1: A Combination of SGA and ASGA*

As shown in Fig. 1(a), each of the K subnets is expected to learn the representation of a class. The

outputs of K subnets are sent to a winner-takes-all (WTA) subnet consisting of K units with binary outputs $z_1, \ldots, z_K$. The function of WTA is described by

$$z_k = \begin{cases} 1, & \text{if } \delta_k(\mathbf{x}) = \max\{\delta_i(\mathbf{x}), i = 1, \ldots, K\}, \\ 0, & \text{otherwise.} \end{cases} \tag{9a}$$

The function could be realized in several ways, e.g., by lateral inhibition between the K units[18,19] and by competitive activation.[18,20] Here, we do not intend to discuss it in detail. Instead, we regard WTA subnet in Fig. 1(a) as a functional box which operates in the way described by Eq. (9a).

Among the other K subnets in Fig. 1(a), there are two types. One is called O-subnet as shown in Fig. 1(b), which is used for learning the representations of the own subspace of one class. The other is called C-subnet as shown in Fig. 1(c), which is for learning the representation of the complementary subspace of one class.

As shown in Fig. 1(b), an O-subnet consists of two layers. Its first layer has $p_k \leq n/2$ units. Each unit has three kinds of input signals. One is n-dimensional data which is either $\mathbf{x}^{(0)} = \mathbf{x} = [\xi_1, \ldots, \xi_n]^t$ directly from outside for the first unit or $\mathbf{x}^{(j-1)} = [\xi_1^{(j-1)}, \ldots, \xi_n^{(j-1)}]^t$ from the $(j-1)$th unit for jth unit. The second is the internal feedback signal $-y_j^{(k)}$ from the output of itself which results in Eq. (9b) through the weights $\mathbf{m}_j^{(k)} = [\mu_{1j}^{(k)}, \mu_{2j}^{(k)}, \ldots, \mu_{nj}^{(k)}]^t$:

$$\mathbf{x}^{(j)} = \mathbf{x}^{(j-1)} - 2y_j^{(k)}\mathbf{m}_j^{(k)}$$

$$= [\xi_1^{(j)}, \ldots, \xi_n^{(j)}]^t - 2y_j^{(k)}[\xi_1^{(j-1)}, \ldots, \xi_n^{(j-1)}]^t \tag{9b}$$

$$y_j^{(k)} = \mathbf{x}^t\mathbf{m}_j^{(k)} \quad \text{or} \quad y_j^{(k)} = (\mathbf{x}^{(j-1)})^t\mathbf{m}_j^{(k)} . \tag{9c}$$

The third is a supervise signal $\chi_k(\omega_\mathbf{x})$ which controls the learning in this subnet and will be discussed later. Moreover, each unit has two output signals. One is $-y_j^{(k)}$ for the internal feedback, the other is the U-type signal $y_j^{(k)^2}$ for sending the output to the second layer which is just a unit for summing up the outputs from the $p_k \leq n/2$ first layer units.

The structure of a C-subnet, as shown in Fig. 1(c), is similar to that of an O-subnet except that there is an extra unit which sums up the normal $\|\mathbf{x}\|^2$ into the unit of the second layer, and the output end for sending signals to the second layer is an inverted U-type. As will be indicated later, the main difference of the two types lie in their learning mechanisms.

The whole process of implementing DSPR by the model given in Figs. 1(a)–(c) consists of the following three phases.

4.1.1. *The pre-structuring phases*

Somewhat similar to backpropagation model in which the structure of the net as well as the number of

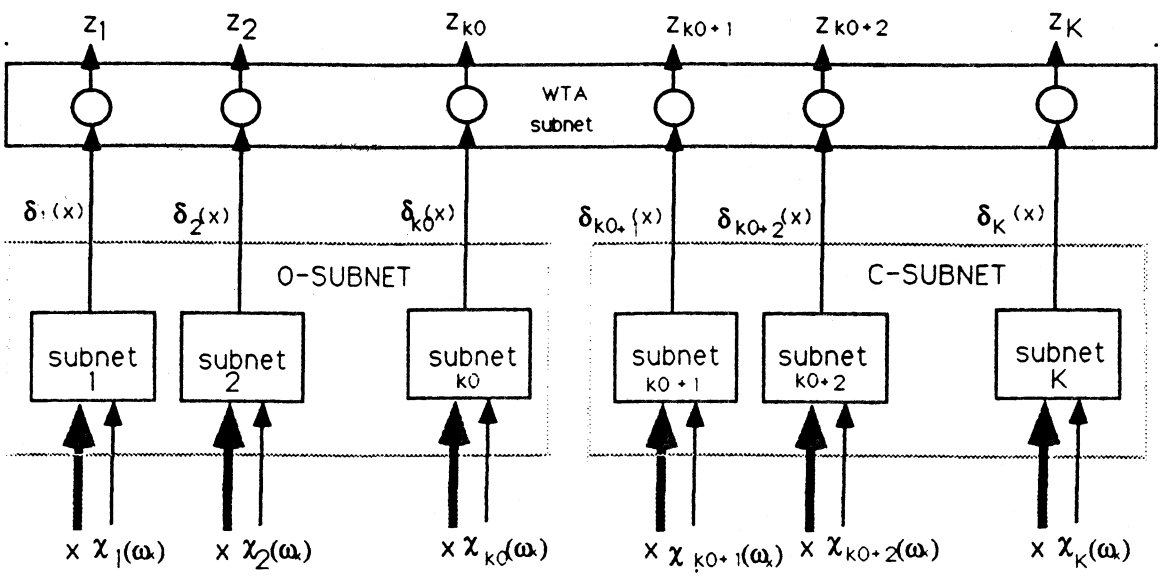


Fig. 1(a). The architecture of neural net models for DSPR based on adaptively extracting orthogonal basis vectors. It is a three-layer net. The first two layers consist of K subnets. The outputs of K subnets are sent to the third layer which is a winner-takes all (WTA) subnet.

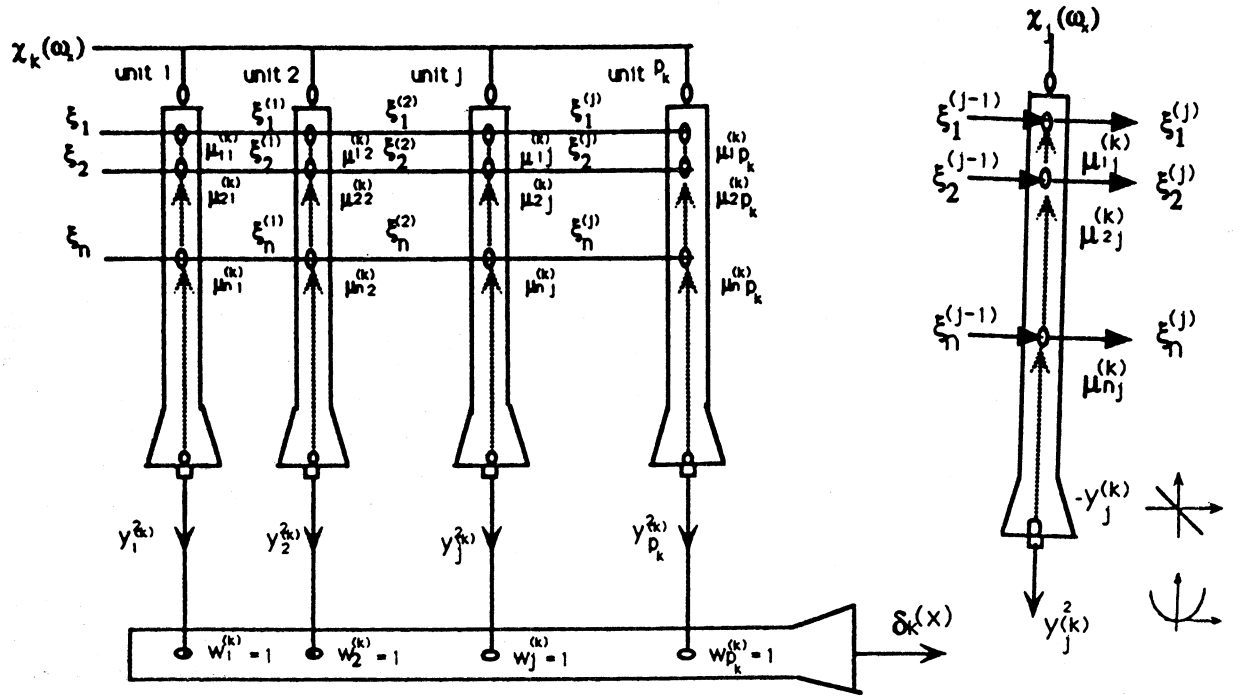


Fig. 1(b). An O-subnet in Model 1. Its first layer consists of $p_k \leq n/2$ units. The details of a unit are shown on the right side. The n-D input signal from the left side is passed to the right side after being subtracted by the internal feedback signal from the output end of the unit and through the connection weights of the units. The signal from the top is a supervise signal which controls the learning. The unit has two output ends. One linear end y_j is for sending the feedback signal, the other U-type nonlinear end y_j^2 for sending the output to the second layer, which is just a unit for summing the outputs from the $p_k \leq n/2$ first layer units.

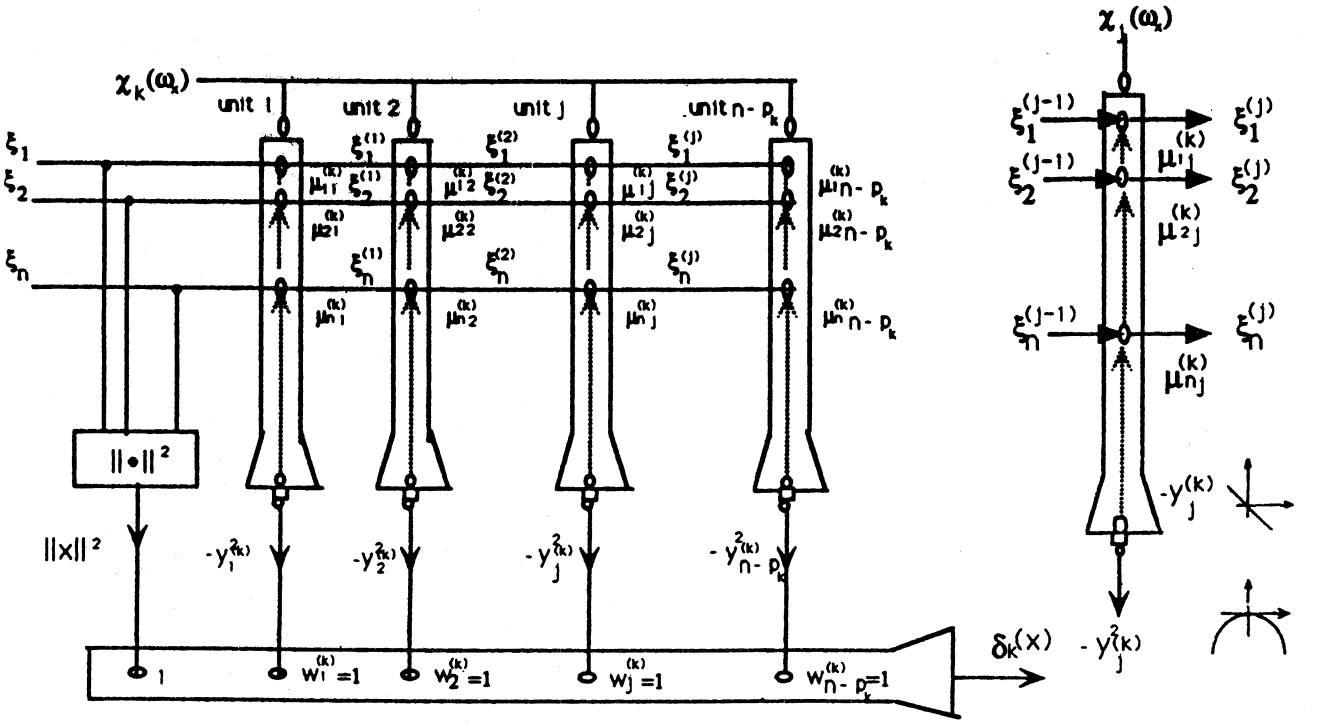


Fig. 1(c). A C-subnet in Model 1. Its first layer consists of $n - p_k \leq n/2$ units. A unit is quite similar to that in Fig. 1(b) except that here the output end for sending signals to the second layer is an inverted U-type. Furthermore, the second layer unit has an additional input $\|x\|^2$.

hidden units are externally predetermined, the structure of the model given in Fig. 1(a) also needs to be predesigned. One needs to know the dimension p_k (or the estimation of its upper bound) of the subspace spanned by each class. An O-subnet with p_k units on its first layer is allocated for learning the representations of the class if p_k (or the estimation of its upper bound) is less than or equal to $n/2$; or, a C-subnet with $n - p_k$ units on its first layer is allocated for learning the representations of the class if p_k (or the estimation of its upper bound) is larger than $n/2$.

4.1.2. *The learning phase*

Each training sample is randomly picked from a training set of K classes. The class label $\omega_{\mathbf{x}}$ of a sample $\mathbf{x}$ is represented to the net in the form of a supervise signal denoted as follows:

$$\chi(\omega_{\mathbf{x}}) = [\chi_1(\omega_{\mathbf{x}}), \chi_2(\omega_{\mathbf{x}}), \ldots, \chi_{\mathbf{K}}(\omega_{\mathbf{x}})] \quad (9d)$$

where $\omega_{\mathbf{x}}$ may be any of the $K + 1$ labels ω_1, $\omega_2, \ldots, \omega_K$ and ϕ, and ϕ is a null label which denotes that the class label of a sample $\mathbf{x}$ is unknown. Each $\chi_k(\omega_{\mathbf{x}})$ is defined by

$$\chi_k(\omega_{\mathbf{x}}) = \begin{cases} 1, & \text{when } \omega_{\mathbf{x}} = k\,; \\ 0, & \text{when } \omega_{\mathbf{x}} \neq k \text{ or } \omega_{\mathbf{x}} = \phi\,. \end{cases} \quad (9e)$$

The net adapts to the input sample $\mathbf{x}$ by modifying the weights $\mu_{ij}^{(k)}$'s. Specifically, the weights in each O-subnet (i.e., when $0 \leq k \leq k_0$) are modified according to the following learning equations [please refer to Fig. 1(b) as well]:

$$\Delta\mathbf{m}_j^{(k)} = \alpha_1 \chi_k(\omega_{\mathbf{x}}) y_j^{(k)} (\mathbf{x}^{(j-1)} - y_j^{(k)} \mathbf{m}_j^{(k)}) \quad (10a)$$

$$y_j^{(k)} = \mathbf{x}^t \mathbf{m}_j^{(k)} \quad \text{or} \quad y_j^{(k)} = (\mathbf{x}^{(j-1)})^t \mathbf{m}_j^{(k)}\,,$$

$$\Delta\mathbf{m}_j^{(k)} = \mathbf{m}_j^{(k)}(t+1) - \mathbf{m}_j^{(k)}(t) \quad (10b)$$

where $0 < \alpha_1 < 1$ is the learning rate which can be appropriately arranged according to the classical results of Robbins and Monro.[21]

Equation (10a) is the same as Oja's SGA[4] when $y_j^{(k)} = \mathbf{x}^t \mathbf{m}_j^{(k)}$ is used in Eq. (10b), since $\mathbf{x}^{(j-1)} = \mathbf{x}^{(j-2)} - 2y_{(j-1)}^{(k)} m_{(j-1)}^{(k)} = \mathbf{x} - 2\sum_{r<j} y_r^{(k)} m_r^{(k)}$ and by putting this into Eq. (10a) one just gets SGA learning rule.[4] As shown in Oja's recent paper,[4] SGA is different from the Generalized Hebbian Algorithm (GHA)[24] in that here the term $\mathbf{x} - 2\sum_{r<j} y_r^{(k)} m_r^{(k)}$ is used instead of using $\mathbf{x} - \sum_{r<j} y_r^{(k)} m_r^{(k)}$ in GHA. SGA can adaptively perform the Gram–Schmidt orthonormalization (GSO) and has a better behavior for extracting the less dominant eigenvectors.

When $y_j^{(k)} = (\mathbf{x}^{(j-1)})^t \mathbf{m}_j^{(k)}$ is used in Eq. (10b), Eq. (10a) is not directly SGA but tends to be the same since $(\mathbf{x}^{j-1})^t \mathbf{m}_j^{(k)} = \mathbf{x}^t \mathbf{m}_j^{(k)}$ when $\mathbf{m}_r^{(k)}$, $r = 1, \ldots, j$ become orthogonal to each other. It follows from Fig. 1(b) that the latter alternative for $y_j^{(k)}$ needs less hard-wirings in the neural net than the former. In fact, considering $\mathbf{x}^{(j-1)}$ as the input of the jth unit, Eq. (10a) is just Oja's constrained Hebbian learning rule for one unit case.[1] That is, SGA is one kind of generalization of Oja's one-unit rule. As proved by Oja,[17] for $t \to \infty$, $\mathbf{m}_j^{(k)}$, $j = 1, 2, \ldots, p_k$ will converge to the eigenvectors of the class correlation matrix $R_i = E[\mathbf{x}\mathbf{x}^t \,|\, \mathbf{x} \in \omega_k]$ corresponding to the p_k largest eigenvalues.

The learning equation for the weights in each C-subnet (i.e., when $k_0 < k \leq K$) is given by:

$$\Delta\mathbf{m}_j^{(k)} = -\alpha_2 \chi_k(\omega_{\mathbf{x}}) y_j^{(k)} \left(\mathbf{x}^{(j-1)} - \frac{y_j^{(k)} \mathbf{m}_j^{(k)}}{\|\mathbf{m}_j^{(k)}\|^2} \right) \quad (11a)$$

where $\mathbf{x}^{(j-1)}$ is still given by Eq. (9b) and $0 < \alpha_2 < 1$ is the learning rate. The equation is obtained from Eq. (10a) by two points of changes. The first point is that the learning in Eq. (10a) is changed into the opposite direction (i.e., add a minus sign in front of α_1). Thus, it becomes a constrained anti-Hebbian learning rule. As shown in our recent paper,[16] it will converge in the direction of the minor (i.e., the one with the smallest eigenvalue) component of input data $\mathbf{x}^{(j-1)}$. However, its magnitude will not converge. To amend this, the second point of change is needed: an explicit normalization term $\|\mathbf{m}_j^{(k)}\|^2$ should be put into Eq. (11a) to guarantee that the magnitude of $\mathbf{m}_j^{(k)}$ is unit. As shown in the paper,[22] $\mathbf{m}_j^{(k)}$ will converge to the minor component of input data $\mathbf{x}^{(j-1)}$. As a result, $m_1^{(k)}$ will converge to the minor component of input data $\mathbf{x}$. $m_2^{(k)}$ will converge to the minor component of input data $\mathbf{x}^{(1)} = \mathbf{x} - y_1^{(k)} m_1^{(k)}$, which is the novel part of $\mathbf{x}$ and is orthogonal to $\mathbf{x}$. That is, $m_2^{(k)}$ will converge to the jth minor component of input data $\mathbf{x}$, and thus, for $t \to \infty$, $\mathbf{m}_j^{(k)}$, $j = 1, 2, \ldots, n - p_k$ will converge to the eigenvectors of class correlation matrix $R_i = E[\mathbf{x}\mathbf{x}^t \,|\, \mathbf{x} \in \omega_k]$ corresponding to the $n - p_k$ smallest eigenvalues. The strict proof could be made in a way similar to that used by Oja[4] or Sanger.[24]

Equation (11a) can be regarded as an anti-Hebbian version of SGA given by Eq. (10a); thus we call it ASGA. Although Eq. (11a) is more complicated than Eq. (10a) and is not local within one connection

weight efficient, the implementation of Eq. (11a) is still local within one unit. In Eq. (11a), an explicit normalization term $\|\mathbf{m}_j^{(k)}\|^2$ is used as a divisor to the term $y_j^{(k)}\mathbf{m}_j^{(k)}$. Another version of ASGA with less explicit normalization was recently proposed by Oja,[4] given by:

$$\Delta\mathbf{m}_j^{(k)} = -\alpha_2\chi_k(\omega_\mathbf{x})[y_j^{(k)}(\mathbf{x}^{(j-1)} - y_j^{(k)}\mathbf{m}_j^{(k)}) + \mathbf{m}_j^{(k)}(1 - \|\mathbf{m}_j^{(k)}\|^2)] \tag{11b}$$

where $\mathbf{x}^{(j)}$ is given by the following Eq. (11c) instead of Eq. (9b):

$$\mathbf{x}^{(j)} = \mathbf{x}^{(j-1)} - \gamma y_j^{(k)}\mathbf{m}_j^{(k)} = [\xi_1^{(j)}, \ldots, \xi_n^{(j)}]^t - \gamma y_j^{(k)}[\xi_1^{(j-1)}, \ldots, \xi_n^{(j-1)}]^t \tag{11c}$$

with $\gamma > \lambda_{q_k}/\lambda_n$, $q_k = n - p_k$, and $\lambda_1 > \lambda_2 > \cdots > \lambda_n > 0$ are the eigenvalues of matrix $R_i = E[\mathbf{x}\mathbf{x}^t/\mathbf{x} \in \omega_k]$.

In Eq. (11b), the term $\mathbf{m}_j^{(k)}(1 - \|\mathbf{m}_j^{(k)}\|^2)$ takes the role of an implicit normalization of $\mathbf{m}_j^{(k)}$. Similar to Eq. (11a), the implementation of Eq. (11b) is still local within one unit although not within one connection weight coefficient. When $\lambda_{q_k} < 1$, it can also be proved that[4] for $t \to \infty$, $\boldsymbol{m}_j^{(k)}$, $j = 1, 2, \ldots, q_k = n - p_k$ will converge to the eigenvectors of class correlation matrix $R_k = E[\mathbf{x}\mathbf{x}^t | \mathbf{x} \in \omega_k]$ corresponding to the $q_k = n - p_k$ smallest eigenvalues.

In summary, after training the net for a long enough period, each O-subnet has let the own subspace representation of a pattern class be learned onto its orthogonalized weight vectors $\mathbf{m}_j^{(k)}$, $j = 1, \ldots, p_k$.

4.1.3. *The classification phase*

When a sample with unknown class label is presented to the net, no learning will take place since $\omega_\mathbf{x} = \phi$ and $\chi_j(\phi) = 0$, $j = 1, \ldots, K$. For each subnet, the outputs $y_j^{(k)} = (\mathbf{x}^{j-1})^t\mathbf{m}_j^{(k)} = \mathbf{x}^t\mathbf{m}_j^{(k)}$ (since $\boldsymbol{m}_r^{(k)}$, $r = 1, \ldots, j$, are orthogonal) of the first layer are summed up to give the output of the second-layer unit

$$\delta_k(\mathbf{x}) = \begin{cases} \sum_{j=1}^{p_k} y_j^{(k)2}, & \text{for } 0 \le k \le k_0; \\ \|\mathbf{x}\|^2 - \sum_{j=1}^{n-p_k} y_j^{(k)^2}, & \text{for } k_0 < k \le K. \end{cases} \tag{12a}$$

As discussed before, after the net has stabilized during the learning period, $\mathbf{m}_j^{(k)}$'s are orthogonal eigenvectors of the class correlation matrix R_k. As a result, Eq. (12a) becomes

$$\delta_k(\mathbf{x}) = \begin{cases} \sum_{j=1}^{p_k} [(\mathbf{m}_j^{(k)})^t\mathbf{x}]^2, & \text{for } 0 \le k \le k_0; \\ \|\mathbf{x}\|^2 - \sum_{j=1}^{n-p_k} [(\mathbf{m}_j^{(k)})^t\mathbf{x}]^2, & \text{for } k_0 < k < K. \end{cases} \tag{12b}$$

The equations are just those given by Eqs. (3c) and (5c).

Finally, the WTA subnet outputs a $z_j = 1$, which means that $\mathbf{x}$ is classified into class ω_j. From Eq. (12a) it is not difficult to see that the function of Eq. (9a) is equivalent to the function of Eq. (6).

So we see from the above discussion that the model given in Fig. 1(a) supplies a neural network way for implementation of dual subspace pattern recognition, and the model is just the counterpart of those conventional approaches like CLAFIC.[10]

4.2. *Model 2: A combination of ALIA and AHALIA*

The architecture of the model is still the same as that given in Fig. 1, but now Figs. 1(b) and 1(c) are replaced by Figs. 2(a) and 2(b), respectively. The main difference between Fig. 1(b) and Fig. 2(a) [as well as Fig. 1(c) and Fig. 2(b)] lies in the fact that for the jth unit in Figs. 2(a) and 2(b) the feedback signal $-y_j^{(k)}$ does not internally and directly go back its input ends as that in Figs. 1(b) and 1(c). Instead, the signal is asymmetrically sent to laterally inhibit the latter units through an additional set of lateral weights $\nu_{ij}^{(k)}$'s which has its own learning rule to be described in the sequel.

In a way similar to that for the first model discussed before, we briefly describe here the whole process of pattern recognition by the present model. First, in the pre-structuring phase, O-subnets and C-subnets are allocated to each pattern class in the same way as that for the first model. Second, in the learning phase, the learning for an O-subnet is made by the following two learning equations:

$$\Delta\mathbf{m}_j^{(k)} = \alpha_1\chi_k(\omega_\mathbf{x})y_j^{(k)}(\mathbf{x} - y_j^{(k)}\mathbf{m}_j^{(k)}) \tag{13a}$$

$$\Delta\nu_{ij}^{(k)} = -\alpha_2\chi_k(\omega_\mathbf{x})y_i^{(k)}y_j^{(k)} \tag{13b}$$

$$y_j^{(k)} = \mathbf{x}^t\mathbf{m}_j^{(k)} - \sum_{i=1}^{j-1}\nu_{ij}^{(k)}y_i^{(k)},$$

$$\Delta\nu_{ij}^{(k)} = \nu_{ij}^{(k)}(t+1) - \nu_{ij}^{(k)}(t) \tag{13c}$$

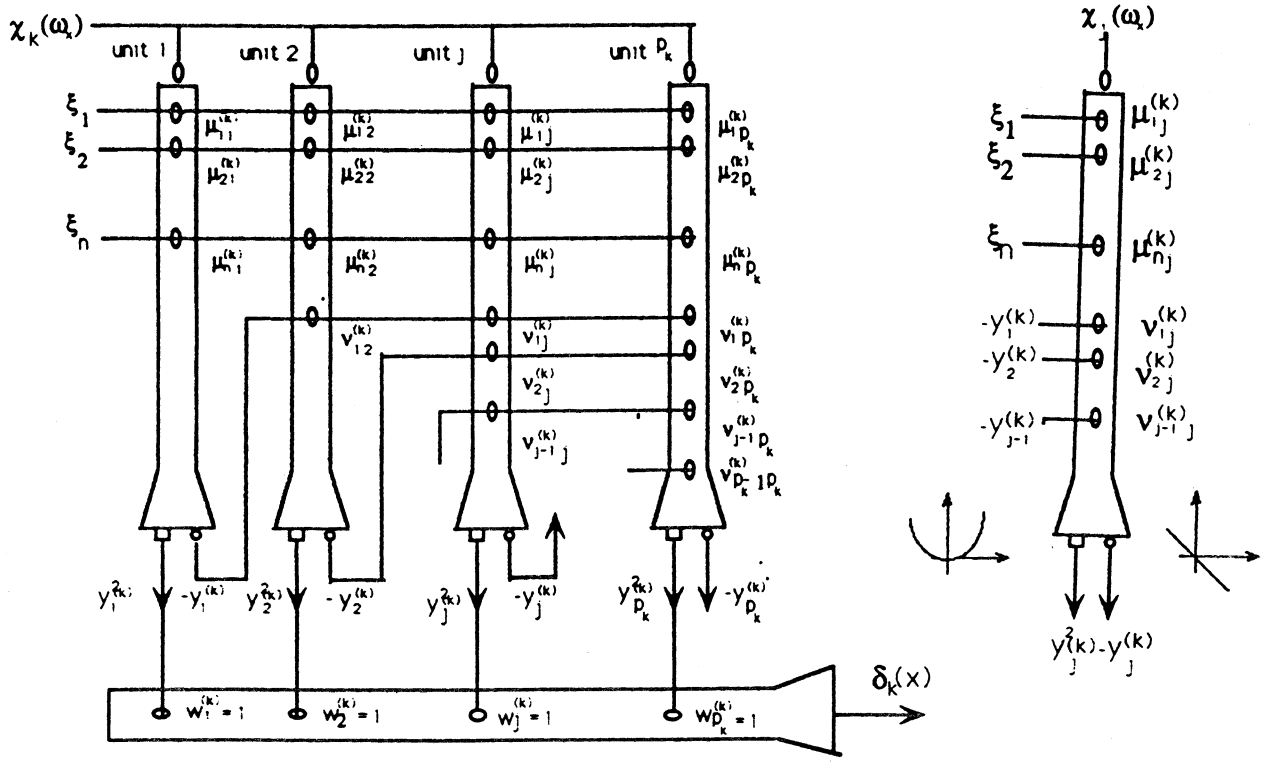


Fig. 2(a). An O-subnet in Model 1. Its main differences from that in Fig. 1(b) consist of two points. First, the n-D input signal from the left side is directly passed to the right side without any modification. Second, the feedback signal from the output end of the unit is not sent to itself but to the latter units as the lateral inhibiting signals through another set of connection weights.

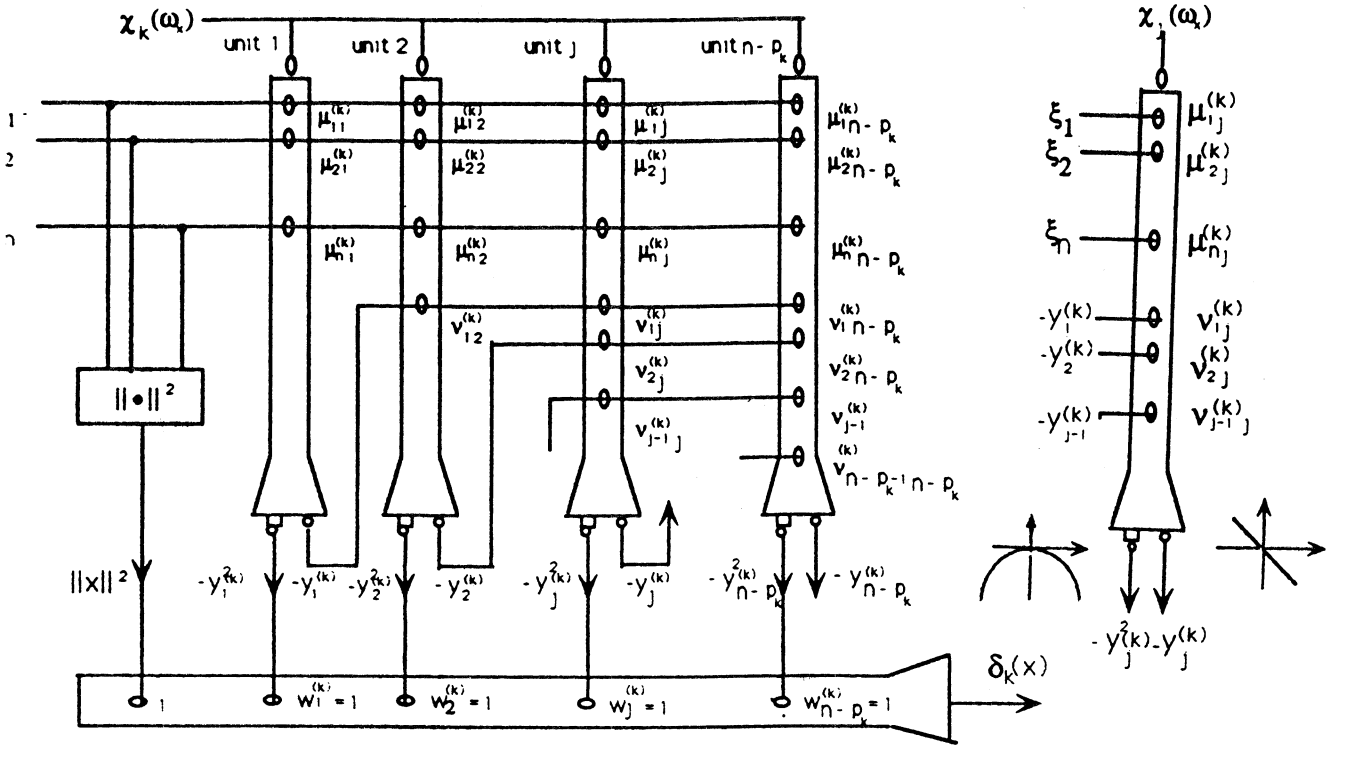


Fig. 2(b). A C-subnet in Model 2. Its main difference from that in Fig. 1(c) is similar to the difference between Fig. 2(a) and Fig. 1(b).

where $0 < \alpha_1, \alpha_2 < 1$ are the learning rates. Equation (13a) is just Oja's one-unit rule and Eq. (13b) is the classical anti-Hebbian learning rule. This combination is proposed by Rubner.[5] Since its key feature is the use of asymmetric lateral inhibition learning (i.e., anti-Hebbian learning) for decorrelating the outputs of different units. In the sequel, we call the combination as Asymmetric Lateral Inhibition Learning Algorithm (or simply denoted by ALIA). It has been proved[5,9] that after learning reaches its equilibrium, all the lateral weights $v_{ij}^{(k)}$'s will vanish, and $\mathbf{m}_j^{(k)}$, $j = 1, 2, \ldots, p_k$ will converge to the eigenvectors of the class correlation matrix $R_i = E[\mathbf{x}\mathbf{x}^t | \mathbf{x} \in \omega_k]$ corresponding to the p_k largest eigenvalues.

For C-subnets, the learning equation Eq. (13a) should be replaced by

$$\Delta \mathbf{m}_j^{(k)} = -\alpha_1 \chi_k(\omega_\mathbf{x}) y_j^{(k)} \left(\mathbf{x} - \frac{y_j^{(k)} \mathbf{m}_j^{(k)}}{\|\mathbf{m}_j^{(k)}\|^2} \right) \quad (14)$$

while Eq. (13b) still remains the same. Equation (14) is just the anti-Hebbian learning version of Oja's one unit rule. As indicated earlier, $\mathbf{m}_j^{(k)}$ will converge to the minor component of input data $\mathbf{x}$. Without Eq. (13b), all the $\mathbf{m}_j^{(k)}$'s will converge to the same minor component (i.e., the one corresponding to the smallest eigenvalue). However, it can be proven in a way similar to that used by Rubner and Hornik *et al.*[5,9] that the decorrelation force produced by Eq. (13b) will finally make all the lateral weights $v_{ij}^{(k)}$'s vanish and $m_j^{(k)}$, $j = 1, 2, \ldots, n - p_k$ converge to the eigenvectors of class correlations matrix $R_i = E[\mathbf{x}\mathbf{x}^t | \mathbf{x} \in \omega_k]$ corresponding to the $n - p_k$ smallest eigenvalues. The combination of Eq. (14) and Eq. (13b) is just the anti-Hebbian learning counterpart of ALIA; thus we briefly denote it by AHALIA.

So again we see that after training the net for a long enough period, each O-subnet has let the own subspace representation of a pattern class be learned onto its orthogonalized weight vectors $\mathbf{m}_j^{(k)}$, $j = 1, \ldots, p_k$. Similarly, each C-subnet lets the complementary subspace representation of a pattern class be learned onto its orthogonalized weight vectors $\mathbf{m}_j^{(k)}$, $j = 1, \ldots, n - p_k$.

Finally, from the results of the above learning phase, and noticing that all the lateral weights $v_{ij}^{(k)}$'s vanish such that $y_j^{(k)} = \mathbf{x}^t \mathbf{m}_j^{(k)} - \sum_{i=1}^{j-1} v_{ij}^{(k)} y_i^{(k)} \mathbf{x}^t \mathbf{m}_j^{(k)}$, we can see that the classification phase for model 2 is the same as that for model 1.

4.3. *Some variants*

Some possible variants are given in Figs. 3(a)–(d). The main difference between these variants and the previously discussed models lies in the following three points:

(a) Instead of both O-subnets and C-subnets, only one type of subnet, e.g., O-subnet, is contained in the variant models.

(b) Each O-subnet, as shown in Fig. 3(b) & (c), consists of n units instead of only p_k units.

(c) For the unit of the second layer in each subnet, its weight is not fixed at constant 1. They are obtained by an adaptive learning, which will be explained later.

This kind of variants has two possible cases, obtained by using the model of Fig. 3(b) or 3(c) as subnets in Fig. 3(a) respectively. For simplicity, we choose only one case, i.e., the model of Fig. 3(c) is used as subnets in Fig. 3(a), for describing the differences of the variants from the originals given Secs. 4.1 and 4.2.

One key feature is that the pre-structuring phase is obviously simplified. Instead of estimating the dimension of the subspace spanned by each class, we simply assign n units for each class and leave the dimension selection problem to the learning phase. The cost of such simplification is that more units are used in each subnet such that the advantage of the dual subspace representation is lost. In fact, the variants given in Figs. 3(a)–(c) are the extensions of the ordinary subspace model given in Oja's paper.[14]

In the learning phase, the adaption made on the weight's $\mu_{ij}^{(k)}$'s and $v_{ij}^{(k)}$'s are still given by Eqs. (13a)

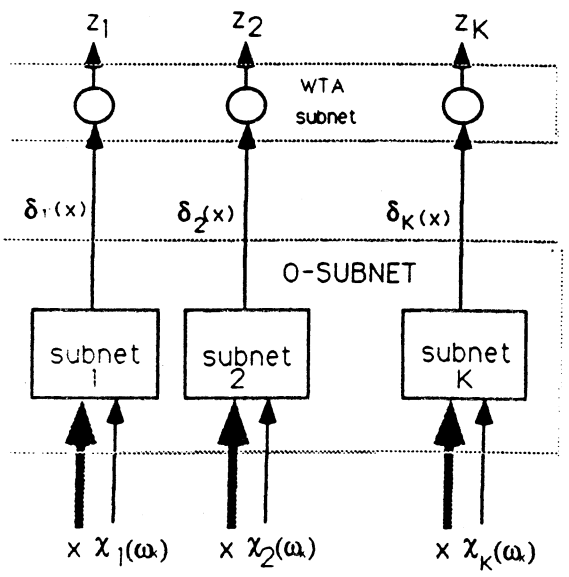


Fig. 3(a). A variant of the model given in Fig. 1(a). Its difference from Fig. 1(a) lies in that it consists of only K O-subnets without any C-subnets.

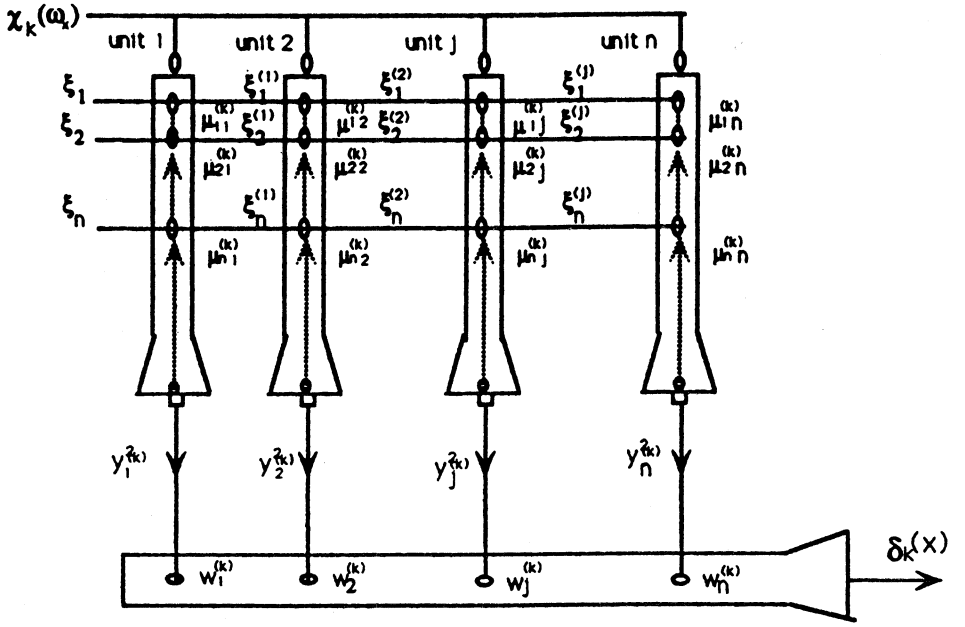


Fig. 3(b). An O-subnet of Model 1 for Fig. 3(a). It is a variant of the O-subnet shown in Fig. 1(b). The different points here are that there are n units in the first layer and the weights of the second layer unit are not fixed at constant 1 but obtained by adaptive learning.

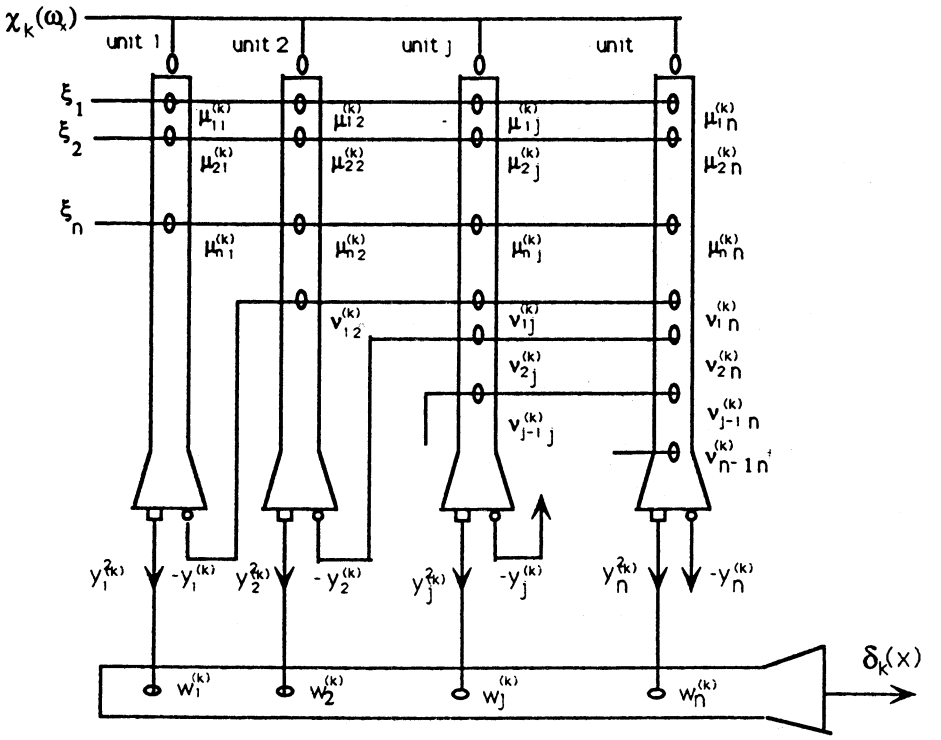


Fig. 3(c). An O-subnet of Model 2 for Fig. 3(a). It is a variant of the O-subnet shown in Fig. 2(a). The differences between Fig. 3(c) and Fig. 2(a) are similar to those between Fig. 3(b) to Fig. 1(b).

and (13b). Besides, there is an additional learning occurring on the weights $\mathbf{w}_j^{(k)}$ of the second-layer unit in each subnet. These weights are modified by the following learning equation:

$$\Delta \mathbf{w}^{(k)} = \alpha_3(\mathbf{y}^{(k)} - \mathbf{w}) \tag{15a}$$

with

$$\mathbf{w}^{(k)} = [w_1^{(k)}, \ldots, w_n^{(k)}]^t, \quad \mathbf{y}^{(k)} = [y_1^{(k)^2}, \ldots, y_n^{(k)^2}]^t$$

where $0 < \alpha_3 < 1$ is a learning rate. This equation is just that used in Kohonen's self-organizing map.[23] As $t \to \infty$, $\mathbf{w}^{(k)}$ tends to $E\mathbf{y}^{(k)} = [E(y_1^{(k)^2}), \ldots, E(y_n^{(k)^2})]^t$. Since $\mathbf{m}_j^{(k)}$, $j = 1, \ldots, n$ will converge to the eigenvectors of the class correlation matrix $R_k = E[\mathbf{x}\mathbf{x}^t]$, we know that $Ey_j^{(k)^2}$ tends to $(\mathbf{m}_j^{(k)})^t R_k \mathbf{m}_j^{(k)}$ which, in turn, tends to the eigenvalue $\lambda_j^{(k)} \geq 0$ corresponding to the jth eigenvector of R_k. As a result, we see from Fig. 3(c) that after training for a long enough time, the output of each subnet is given by

$$\delta_k(\mathbf{x}) = \sum_{j=1}^{n} w_j^{(k)} y_j^{(k)^2} \to \sum_{j=1}^{n} \lambda_j^{(k)} y_j^{(k)^2}. \tag{15b}$$

When the real dimension of subspace spanned by the class k is $p_k < n$, the $n - p_k$ eigenvalues of R_k should be quite small[17]: thus $\sum_{j=1}^{n} \lambda_j^{(k)} y_j^{(k)^2}$ could be regarded as an approximation of $\sum_{j=1}^{p_k} \lambda_j^{(k)} y_j^{(k)^2}$, where $\lambda_j^{(k)} > 0$ are p_k largest eigenvalues of R_k, and $y_j^{(k)^2}$ are outputs of the units with weight vectors being corresponding eigenvectors. Furthermore, $\sum_{j=1}^{p_k} \lambda_j^{(k)} y_j^{(k)2}$ could be taken as an approximation of that given by Eq. (3c), especially when these $\lambda_j^{(k)}$'s are near 1. So, we see that Kohonen learning based on the weights $w_j^{(k)}$'s performs the task of automatically selecting the basis vectors of each subspace.

5. Some Computer Simulations

5.1. *The results on a two-class problem*

Our simulations were made on a data set of two classes in R^3 space. To visualize the distributions of the two classes, we projected these populations onto the x–y plane, x–z plane and y–z plane, respectively, and showed them in Figs. 4(a)–(c). The data set consists of 400 sample points, 200 points were randomly chosen as the training set and the other 200 points as the test set. We can observe from Figs. 4(a)–(c) that for all the three projections, the

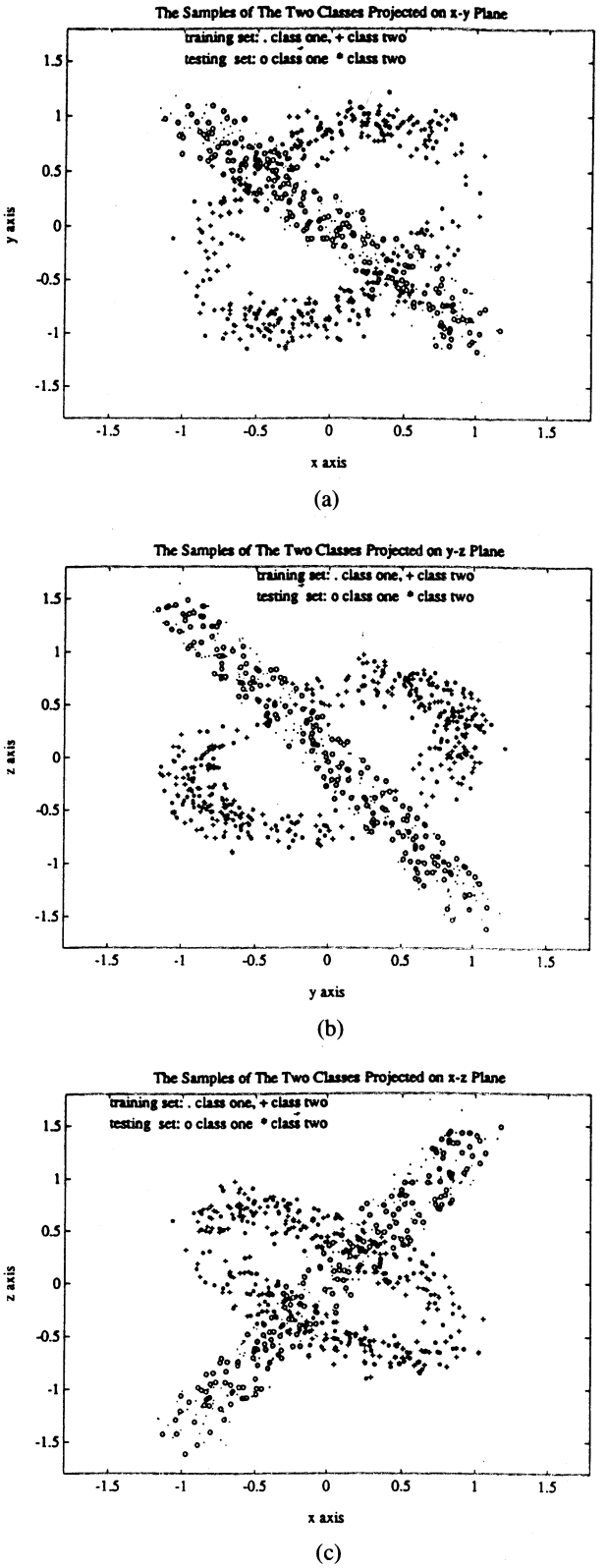


Fig. 4. Data samples of two classes, projected on each of the coordinate planes, where symbols "." and "o" denote respectively a training-sample point and a testing-sample point of class one, symbols "+" and "*" denote respectively a training-sample point and a testing-sample point of class two.

samples of class one located on a line-segment bank and the samples of class two located on an ellipse bank. Furthermore, the two banks overlap each other and have the same mean values located at the origin of each coordinate plane. In fact, in 3D space, the samples of class two are located on an elliptic ring which centers at the origin of the 3D coordinate system and its longest elliptic axis is along the direction $(-1, 1, 0)$, the plane of the two elliptic axes intersects the $x-y$ plane with an acute angle (around 30 degrees); while the samples of class one are located on a straight bar which penetrates the origin of the 3D coordinate system and is perpendicular to the elliptic plane.

Since the two classes have the same mean values, they are difficult to classify by the perceptron or parametric Bayes classifier with Gaussian assumption on populations. Here, we use the neural net model proposed in Sec. 4.1 and Fig. 1 to solve the problem.

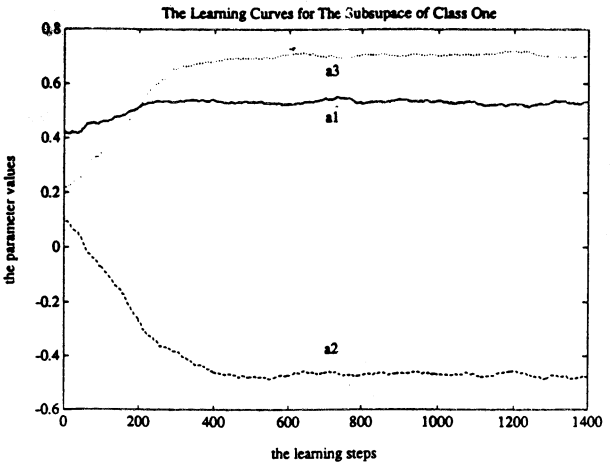
From Figs. 4(a)–(c) or the eigenvalues of the covariance matrix of two classes, it is known that the dimensions of the own subspaces of class one and class two are respectively $p_1 = 1$ and $p_2 = 2$. Thus by dual subspace representation, we can use a vector $\mathbf{m}_1$ to represent class one through representing its own subspace and a vector $\boldsymbol{m}_2$ (since $q_2 = 3 - p_2 = 1$) to represent class two through representing its complementary subspace. We initialized $\mathbf{m}_1$, $\mathbf{m}_2$ by letting the three components of each basis vector be random numbers uniformly distributed on the [0, 1] interval, and then trained them by learning rule Eqs. (10) and (11) with learning rates $\alpha_1 = \alpha_2 = 0.01$. The experiments turned out that the learning converges very fast regardless of the initial values. An example of the learning process are shown in Figs. 5(a), (b). The confusion matrices given in Table I and Table II show the classification performances after 400 learning steps and 900 learning steps, respectively.

5.2. *Some comparisons with backpropagation approach*

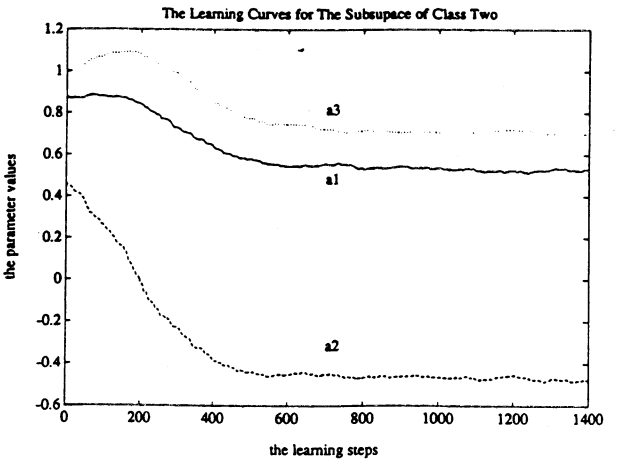
A backpropagation (BP) approach with a three-layer forward net has also been tested on the above data set. Its input layer has three nodes for the three components of the 3D input vector, its output layer has two nodes with each one for representing a pattern class, and its hidden layer has n_h nodes. We have tried several cases of $n_h = 2$, 10, 20, 40, respectively.

For each unit, its output is given by

$$o_j^{(r+1)} = \frac{1}{1 + e^{-y_j^{(r+1)}}}, \qquad y_j^{(r+1)} = \sum_i w_{ij} o_j^{(r)} + \theta_j$$



(a)



(b)

Fig. 5. The learning process of the basis vectors in dual subspace representation for (a) basis vector $\mathbf{m}_1 = [a_1^{(1)}, a_2^{(1)}, a_3^{(1)}]^t$ of the own subspace for representing *class one*. (b) basis vector $\mathbf{m}_2 = [a_1^{(2)}, a_2^{(2)}, a_3^{(2)}]^t$ of the complementary subspace for representing *class two*.

Table I. The confusion matrix after 400 learning steps.

	Training Set		Testing Set	
	class 1	class 2	class 1	class 2
class 1	93%	7%	93.5%	6.5%
class 2	0%	100%	0%	100%

Table II. The confusion matrix after 900 learning steps.

	Training Set		Testing Set	
	class 1	class 2	class 1	class 2
class 1	93.5%	6.5%	93.5%	6.5%
class 2	0%	100%	0%	100%

where $r = 0, 1, 2$ denote the input, hidden and output layer, respectively. The learning is a pure gradient descent without any other modification; the learning rate is fixed at 0.1 for each descent update on every unit. The learning is of the on-line (or adaptive) type, i.e., at each learning step, one input sample $\mathbf{x}$ is randomly selected from the training set with equal chance to be from either of the two classes, then the current (i.e., not averaged with the earlier steps) error

$$E = \sum_{j=1}^{2} [t_j(\mathbf{x}) - o_j^{(2)}(\mathbf{x})]^2$$

is minimized in the gradient descent through BP.

The classification results by BP are given in the following Table II and Table IV.

In the following, we give some observations about the performances on the data set given in Sec. 5.1 by backpropagation (BP) and our neural model of Fig. 1(a) (for convenience, we denote it by NDSM, i.e., neural dual subspace method).

(1) It follows from Tables I, II, III and IV that the classification rates of NDSM are obviously better than those of BP, e.g., on the testing set, NDSM could easily obtain the average recognition rate $0.5 \times 93.5\% + 0.5 \times 100\% = 96.75\%$, while the best case (i.e., $n_h = 20$) of BP could only reach the average recognition rate $0.5 \times 86\% + 0.5 \times 98\% = 92\%$.

(2) The computational costs of BP are significantly greater than those of NDSM. On the storage aspect, the main storage requirement of BP is for weight vectors, the outputs and the errors of the hidden units. It approximately needs the storage for $S_{BP} = 3 \times n_h + 2n_h + 2 \times n_h$ real numbers, specifically, $S_{BP} = 140$ for the best performance case $n_h = 20$; while NDSM needs only the storages for two base vectors, i.e., $S_{NDSM} = 2 \times 3 = 6$ real number, thus we have $S_{BP}/S_{NDSM} = 23.3$. As regards computing time, in the best performance case (in Table IV at $n_h = 20$) BP spent $nr_{BP} = 9\ 130\ 018$ flops on training and $nt_{BP} = 248\ 123$ flops on testing, while in Table II with a better classification rate, NDSM spent only $nr_{NDSM} = 54\ 579$ flops on training and $nt_{NDSM} = 37\ 644$ on testing. Thus we have $nr_{BP}/nr_{NDSM} = 167.28$ and $nt_{BP}/nt_{NDSM} = 6.59$, where the number of flops are counted by MATHWORK Inc.'s PRO-MATLAB software, by which one addition (subtraction) or one multiplication (division) involving a real number is defined as one flop.

(3) NDSM is quite robust to the selection of those predefined parameters such as the learning rate, α, and the initial values of bases vectors. For example, let α vary from 0.01 to 0.2 and for different initializations, the classification rates obtained in Table I and Table II

Table III. The confusion matrices after 5000 learning steps by BP.

	$n_h = 2$		$n_h = 10$		$n_h = 20$		$n_h = 40$	
	c_1	c_2	c_1	c_2	c_1	c_2	c_1	c_2
Training								
class 1	100%	0%	83.5%	16.5%	64%	36%	29%	71%
class 2	100%	0%	6.5%	93.5%	15%	85%	5.5%	94.5%
Testing								
class 1	100%	0%	82%	18%	61%	39%	26%	74%
class 2	100%	0%	10%	90%	12.5%	87.5%	5.5%	94.5%

Table IV. The confusion matrices after 11 000 learning steps by BP.

	$n_h = 2$		$n_h = 10$		$n_h = 20$		$n_h = 40$	
	c_1	c_2	c_1	c_2	c_1	c_2	c_1	c_2
Training								
class 1	0%	100%	89.5%	10.5%	86%	14%	27.5%	72.5%
class 2	0%	100%	5%	95%	2%	98%	3.5%	96.5%
Testing								
class 1	0%	100%	89%	11%	86%	14%	24%	76%
class 2	0%	100%	6%	94%	2%	98%	4.5%	95.5%

remain unchanged. The training time also remains nearly the same because of fast convergence in NDSM learning. However, BP is sensitive to the initialization of weights, especially to the pre-selected learning rate α. For instance, when a learning rate of $\alpha = 0.05$ was used, even after 11 000 learning steps, in our simulations of all the cases that $n_h = 2$, 10, 20, 40 BP always reached unacceptable average recognition rates (around 50%–60%). Usually, to choose an appropriate learning rate and the initial weights, BP needs a lot of extra computational costs for many "try-and-see" trials.

(4) From Table III and Table IV, we see that the best performance is achieved when the number of hidden units is around 10–20. For inappropriate numbers (e.g., $n_h = 2$, $n_h = 40$), we get very bad result even after training the net for a long time. So, similar to NDSM which needs to determine the subspace dimension of each class, BP needs to determine the dimension of the hidden layer. The difference is that for NDSM we at least have a simple formula given in Sec. 3.2 to tackle the task, while for the BP the job is again very complicated and expensive.

(5) Finally, it should be pointed out that BP is a general approach which is widely applicable to many kinds of practical problems, while the subspace methods (including the dual subspace methods proposed in this paper) are limited to some special kinds of practical problems as indicated by Oja.[13,17] So, NDSM could only outperform BP in those problems. We will give some new results for extending the applicable range of subspace methods elsewhere.

6. Conclusions

For data or pattern representation, it is conventionally regarded that principal components are important while minor components are unimportant or are noises. However, there exist cases in which the minor components are of the same importance as principal components. In this paper, we have observed that the minor components have the same representation abilities as the principal components. Based on this observation, we proposed a new subspace representation form—the dual subspace representation, which uses both principal components and minor components for expressing pattern classes. With this new representation, we could usually save much computing time and storage. The nice thing is that all the techniques (including LSM) of the conventional subspace method could be adapted to this new form with small changes. Furthermore, we proposed two neural-net models for implementing dual subspace pattern recognition in an adaptive way by using some combinations of extended Hebbian and/or anti-Hebbian learning rules. The results of computer simulations given in Sec. 5 have demonstrated that our method may provide a new tool for adaptively solving the problems of subspace pattern recognition.

Appendix

The work was supported partly by Tekes Grant 4196 under the Finsoft project (Finland), and partly by the Natural Sciences and Engineering Research Council of Canada and the National Networks of Centers of Excellence program of Canada.

References

1. E. Oja, "A simplified neuron model as a principal component analyzer," *J. Math. Biol.* **16**, 267–273 (1982).
2. P. Foldiak, "Adaptive network for optimal linear feature extraction," *Proc. IEEE International Conf. Neural Networks*, Washington D.C., June 1989, Vol. I, pp. 401–405.
3. J. Rubner and P. Tavan, "A self-organizing network for principal-component analysis," *Europhys. Lett.* **10**, 693–689 (1989).
4. E. Oja, "Principal components, minor components, and linear neural networks," submitted, 1991.
5. J. Rubner and K. Schulten, "Development of feature detectors by self-organization," *Biol. Cybern.* **62**, 193–199 (1990).
6. P. Baldi and K. Hornik, "Neural networks and principal component analysis: Learning from examples without local minima," *Neural Networks* **2**, 52–58 (1989).
7. S. Y. Kung, "Neural networks for extracting constrained principal components," to appear in *IEEE Trans. Neural Networks*.
8. P. Baldi and K. Hornik, "Backpropagation and unsupervised learning in linear networks," in *Backpropagation: Theory, Architectures and Applications*, eds. Y. Chauvin and D. E. Rumelhart (Erlbaum Associates, 1991).
9. K. Hornik and C. M. Kuan, "Convergence analysis of local feature extraction algorithms," *Neural Networks*, to appear.
10. S. Watanabe and N. Pakvasa, "Subspace method of pattern recognition," *Proc. 1st Int. J. Conf. Pattern Recognition*, Washington D.C., Oct. 30–Nov. 1, 1973, pp. 25–32.
11. T. Kohonen *et al.*, "Spectral classification of phenomes by learning subspaces," *Proc. 1979 IEEE Int. Conf. ASSP*, Washington D.C., April 1979, pp. 97–100.
12. E. Oja and M. Kuusela, "The ALSM algorithm—An improved subspace method of classification," *Pattern Recognition* **16**, 421–427 (1983).
13. E. Oja and T. Kohonen, "The subspace learning algorithm as a formalism for pattern recognition and neural networks," *Proc. 1988 IEEE Int. conf. Neural Net-*

works, San Diego, July 1988, pp. 277–284.

14. E. Oja, "Neural networks, principal components, and subspaces," *Int. J. Neural Systems* **1**, 61–68 (1989).
15. J. Karhunen, "On the recursive estimation of the eigenvectors of correlation type matrices," Lic. Tech. Thesis, Helsinki University of Technology, 1982.
16. L. Xu, E. Oja and C. Y. Suen, "Method Hebbian learning for curve and surface fitting," submitted to *Neural Networks*, 1991.
17. E. Oja, *Subspace Methods of Pattern Recognition* (Research Studies Press, Letchworth, UK, 1983).
18. S. Grossberg, "A theory of brain memory: Self-organization and performance of sensory-motor codes, maps, and plans," in *Progress in Theoretical Biology*, Vol. 5, ed. R. Rosen and F. Snell (Academic, New York, 1987).
19. R. P. Lippman, "An introduction to computing with neural nets," *IEEE ASSP Magazine*, Apr. 1987, 4–23.
20. J. A. Reggia, 'Methods for deriving competitive activation," *Proc. 1989 International Joint Conference on Neural Networks*, Vol. I, 1989, pp. 357.
21. H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math, Stat.* **22**, 400–407 (1951).
22. E. Oja and J. Karhunen, "On stochastic approximation of eigenvectors and eigenvalues of the expectation of a random matrix," *J. Math. Anal. Appl.* **106**, 69–84 (1985).
23. T. Kohonen, *Self-organization and Associative Memory* (Springer, Berlin, 1988).
24. T. D. Sanger, "Optimal unsupervised learning in a single-layer linear feedforward neural network," *Neural Networks* **2**, 459–473 (1989).