(2)

Report DARPA/RD/0992

FINAL REPORT

NEURAL NETS FOR SCENE ANALYSIS

David Casasent (PI)
Carnegie Mellon University
Center for Excellence in Optical Data Processing
Department of Electrical and Computer Engineering
Pittsburgh, PA 15213
(412) 268-2464

September 1992
Final Technical Report
Contract DAAH01-89-C-0418
31 March 1989 - 30 September 1992

UNCLASSIFIED

Sponsored by:
Defense Advanced Research Projects Agency (DoD)
Defense Sciences Office
ARPA Order Nr. 6671
Issued by U.S. Army Missile Command
Contract DAAH01-89-C-0418

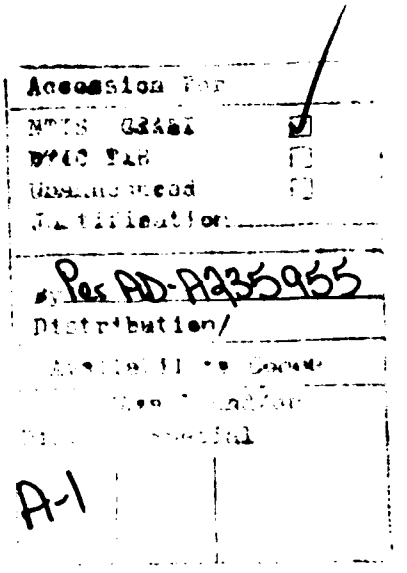
DISCLAIMER

"The view and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government."

NEURAL NETS FOR SCENE ANALYSIS

CHAPTER 1: INTRODUCTION

This project involved various new optical and digital neural net techniques for scene analysis. The original neural net concept was the adaptive clustering neural net (ACNN). This is detailed in Chapter 2. Our original associative processor concept was the Ho-Kashyap neural net. This is detailed in Chapter 3. Our overview of how neural nets should be used in scene analysis is detailed in Chapter 4. This also includes an overview of our two new higher order neural nets. Our new PQNN neural net (which produces higher-order decision surfaces much more efficiently than other neural nets) is noted in Chapter 5. To achieve high performance on systems with components with analog accuracy and various nonidealities, we developed a new algorithm and technique discussed in Chapter 6. We have fabricated our optical laboratory neural net and tested it on several different case studies and achieved excellent results as noted in Chapter 7.



Adaptive-clustering optical neural net

David P. Casasent and Etienne Barnard

Pattern recognition techniques (for clustering and linear discriminant function selection) are combined with neural net methods (that provide an automated method to combine linear discriminant functions into piecewise linear discriminant surfaces). The resulting adaptive-clustering neural net is suitable for optical implementation and has certain desirable properties in comparison with other neural nets. Simulation results are provided.

I. Introduction

Artificial neural networks have received much recent attention[1-3] and various optical realizations[4,5] of the classic backpropagation neural network[6] have been suggested. Various other optical neural network architectures have been described[7-9] and some[10-13] have been demonstrated conceptually. In this paper we distinguish between optimization and adaptive learning neural networks (Sec. II) and we discuss various neural net issues as background. We then advance a new adaptive-clustering neural network (ACNN) in Sec. III. Simulation results (performed on a Hecht-Nielsen Corporation electronic neural network) are then presented (Sec. IV), optical realizations of the ACNN are discussed (Sec. V) and a summary is advanced (Sec. VI). This ACNN uses a new learning algorithm that combines standard pattern recognition techniques and neural net concepts to arrive at a new and quite useful method for neural network synthesis that can be achieved optically with attractive results and potential.

II. Artificial Neural Networks

We distinguish between two main classes of neural networks[14,15]: optimization neural nets and adaptive learning neural nets. Optimization neural nets are well understood and their basic theory is well established.[16,17] Associative processors are another class of neural networks[16,18-21] that are also well understood. In this paper we consider adaptive learning neural nets. The major advantage of a neural net in multiclass pattern recognition is its ability to compute nonlinear decision surfaces (typically combinations of linear decision surfaces) for complex multiclass decision problems. In fact, many neural net classifiers can create decision boundaries of arbitrary shape. Our proposed neural net uses this feature of neural nets in conjunction with initial weights selected using class prototypes of clusters—hence we refer to this as an adaptive-clustering neural net. It employs a three-layered architecture, consisting of input, hidden, and output layers with interconnections between the input and hidden layers, and between the hidden and output layers.

A. Neuron Representation Spaces and Dimensionality

To maintain a reasonable number of input (P_1) neurons, we recommend[14,15] that the neuron representation space be an appropriate feature space. For image recognition applications, the feature space should not be pixel-based. Other feature spaces have the additional advantage that they can be made invariant to transformations such as in-plane rotations. This greatly reduces the number of training images required (i.e., we need not train on transformed versions of the objects to be identified). For an M-dimensional feature space, we use $M + 1$ input neurons. The additional neuron is used to incorporate the threshold of the hidden layer neurons into the input vector with the state of this neuron set to unity. We now detail this. A linear discriminant function (LDF) in a feature space described by feature vectors $\mathbf{x}$ can be written as

$$g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0, \tag{1}$$

where $\mathbf{w}$ defines the orientation of the linear decision boundary and w_0 defines its offset or location. When decisions depend on whether $g \gtrless 0$, then $-w_0$ is the threshold for the vector–inner product (VIP) $\mathbf{w}^t\mathbf{x}$. By adding an additional 1 to the feature vector $\mathbf{x}$ to produce $\mathbf{y}$, we include w_0 in $\mathbf{w}$ and we can now write Eq. (1) as

$$g(\mathbf{x}) = \mathbf{w}^t\mathbf{y}. \tag{2}$$

The authors are with Carnegie Mellon University, Department of Electrical & Computer Engineering, Center for Excellence in Optical Data Processing, Pittsburgh, Pennsylvania 15213-3890.

Received 28 August 1989.
0003-6935/90/172603-13$02.00/0.

The number of neurons in layer-two (hidden layer) is generally chosen empirically. The number of hidden layer neurons determines the complexity of the decision surface. Thus, too few neurons lead to poor classification performance, since a decision surface of complexity sufficient to separate the various classes cannot be created. In most neural nets, the use of too many hidden neurons is wasteful of resources and leads to poor generalization. By this we mean that the decision surfaces are adapted to the peculiarities of the training set.

Local minima are a frequent topic of discussion associated with the number of hidden neurons used. A local minimum is a value of the energy function that is a minimum in a local region, rather than being a global minimum. In training a backpropagation (BP) neural net,[6] the initial state of the hidden layer neurons is random and a given error rate and some energy is obtained. When training is repeated with different initial hidden neuron states, if a different error rate results, a local minimum exists. One must vary the number of hidden neurons and retrain with different initial conditions to empirically determine the number of hidden neurons. The presence of such variables results in long training times for neural nets (as various numbers of layer-two neurons and various starting conditions are tried) and it can result in a neural net that cannot easily be generalized to test data.

Local minima occur when hidden neurons become redundant during training (e.g., two of the N hidden neurons encode decision boundaries that lie very close to one another). If each neuron encoded a distinct decision boundary, a lower error rate would result (if the number of neurons were too few). When the number of distinct hidden neurons is sufficient (equal to or greater than the minimum required), there is no effect on classification performance, since sufficiently complex decision surfaces can be created despite redundancies in the hidden neurons. Thus, in this case local minima are not of concern. Many researchers have found that extensive methods to produce 100% classification on training data are not merited, since test set performance often does not reflect such improved training set results. Recent work[22,23] on the choice of the number of hidden neurons has concentrated on the case when the training samples are in random positions in the feature space, which is almost never the case in real pattern recognition problems.

Thus, although local minima are not of major concern, an alternate technique to determine the number of hidden neurons with significantly reduced effort is a significant concern. Our new neural net addresses this issue by an organized procedure that selects the number of hidden neurons based on the number of clusters present in the multiclass data to be separated (as detailed in Sec. III).

The number of neuron layers used is another variable. For BP, it has been shown[24,25] that any decision surface can be approximated to arbitrary accuracy with a three-layer neural net. Four-layer neural nets can also produce any such decision surface, but they are harder to train (since the Hessian of the criterion function with respect to the weights is more ill-conditioned when more layers are used) and they generally introduce more parameters that must be empirically selected. Since our neural net also approximates any such decision boundary with three layers, we restrict attention to a three-layer neural net.

The number of output-layer neurons equal the number of classes.

B. Criterion or Error Functions

One of the most popular adaptive learning neural nets is backpropagation (BP).[6] The problems with this neural net are that it requires a large training set and long training time, and does not necessarily converge to the best minimum. Backpropagation is an example of a neural net which is trained by the minimization of an error or criterion function. The form of the error function that is minimized for such nets can affect performance and training time (e.g., the error function with the best error rate is often the one for which it is most difficult to reach a minimum error[26]). Standard BP uses an error function based on a sigmoid transfer function, while our ACNN uses the perceptron error function in training. We recently provided[26] a comparison of various error or criterion functions. It was shown that, in general, the use of a perceptron criterion function provides faster convergence with comparable error rates P_e to those obtained with the more popular sigmoid criterion function. The error function choice is not of major concern in the performance of BP and our ACNN (it is included to note the differences between BP and ACNN and because the criterion function used specifies the type of linear classifier employed, as we detail in Sec. III).

C. Update Algorithm

One reason for the slow convergence of BP is that a gradient-descent (delta rule) algorithm is often used to update the weights in training. Our ACNN uses a conjugate-gradient algorithm[27] for weight update since it is faster and does not require the empirical choice of parameters such as the learning rate and momentum.[26,28] In conjugate-gradient updating, all of the training set data are fed to the system (once) and then the weights are updated. Conversely, with gradient descent the weights can be updated after the presentation of each sample in the training set. A batch type of gradient-descent algorithm can also be used, with weights updated only after all training data have been presented to the system once. Generally, batch gradient descent has the slowest convergence (since the parameters cannot be updated and selected at different steps). Sequential (nonbatch) gradient descent generally performs better than batch gradient descent, since it makes more steps toward the solution (in one presentation of the training set of data). However, selection of its parameters is empirical and we have found that conjugate-gradient optimization performs better. We attribute this to the fact that conju-

gate-gradient optimization adapts the learning parameters in a sensible way, whereas these parameters are kept fixed or adapted heuristically for gradient descent.

In difficult multiclass decision problems we have found conjugate-gradient training to be much more efficient than gradient descent. With neural net hardware and software (such as the Hecht-Nielsen Corp. AZP which we use) conjugate-gradient optimization is very attractive. In our comparisons of BP and the ACNN we use the same conjugate-gradient algorithm to update the weights.

D. Initial Weights

Another reason for the long training time for BP is that the initial weights are chosen arbitrarily. In our ACNN algorithm, the initial weights are set using pattern recognition techniques and then they are refined using neural network techniques. This is a major reason for the improved performance of our ACNN. We have tested BP using initial weights chosen from clustering techniques similar to those used for the initial weights of the ACNN. We found[29] negligible improvement in training time and worse performance in some cases. We attribute this to the fact that BP can sometimes use hidden neurons in more sophisticated ways than is the case in the hidden layer of our ACNN and that this cannot be achieved when a preset weight choice is used.

This present section was intended to highlight issues associated with neural networks and to note differences between our algorithm and the more extensively tested and analyzed BP algorithm.

III. Adaptive Clustering Neural Net (ACNN) Training Algorithm

Our three-layer ACNN is shown in Fig. 1. It is similar to the standard multilayer perceptron. We now detail its design and use for multiclass pattern recognition. The input (P_1) neurons are analog and represent a feature space which can be of low dimensionality (we add an additional feature which is always kept at unity to adapt the threshold of the hidden neurons as well). The hidden layer neurons at P_2 correspond to clusters in feature space, with several clusters (neurons) used for each class in a multiclass application. The P_1–P_2 weights are used to assign an input to a cluster. We typically use two to five clusters per class. The layer-two neurons are binary and (in testing) the P_2 neuron with the largest input activity fires and denotes the cluster to which the input belongs. During training the P_1–P_2 weights adapt as we will detail (we employ a conjugate-gradient algorithm) and thus refine our initial weight estimates. The hidden layer-to-output weights are fixed (all are either zero or one) and perform the mapping of the P_2 clusters to one of the classes (with one P_3 neuron assigned per class of data). Thus, we initially assign several layer-two cluster neurons to each class and use fixed P_2–P_3 weights to assign each P_2 cluster to a final class (output neuron in P_3). This is attractive and new since it allows us to use standard clustering and pattern recognition techniques to select the initial P_1–P_2 weights (initial LDFs) and new neural net techniques to adapt or refine these weights. We employ a perceptron criterion or error function (this defines our LDFs) rather than a sigmoid error function, since faster convergence with a comparable error rate is obtained.

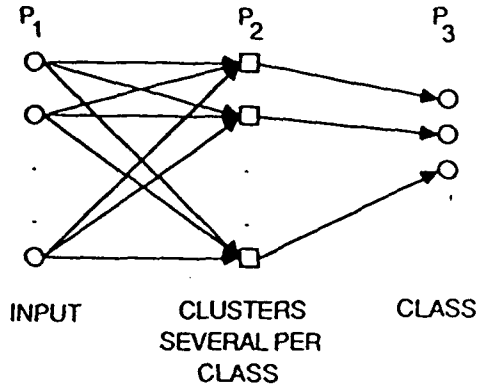


Fig. 1 Adaptive-clustering neural net.

There are no commonly used standard (non-neural net) techniques to obtain piecewise linear decision surfaces for two- or multiclass problems (except nearest-neighbor methods). Because of the importance of neural net techniques in addressing this problem, and since we use nearest-neighbor techniques in selecting our clusters, we briefly review standard multiclass techniques. In a nearest-neighbor classifier, the distance between an input and all training samples is calculated and the input is assigned to the class of the closest training sample. From tests on all training data in each class, the bounds on each class are determined and one can obtain piecewise linear decision surfaces. However, the nearest-neighbor technique is computationally intensive (requiring calculation of the distance to all training samples). Conversely, neural nets have a long training time (which is off-line and of less concern) but their classification times (an on-line requirement) are short. In addition, all training samples must be stored for a nearest-neighbor system and thus storage requirements can be excessive. Finally, nearest-neighbor systems do not perform well when the probability-density functions of the classes overlap significantly. The calculation of the K nearest neighbors is useful here (the input is assigned to the class to which the majority of these K samples belong). However, the selection of K is empirical.

Two other multiclass techniques are Gaussian and linear classifiers. Gaussian classifiers assume that the data in each class are normally distributed and for each class its mean and variance are estimated. To classify an input vector, *a posteriori* probabilities are calculated for each class with Bayes' rule, and the input is assigned to the class with the highest probability. This technique (and all parametric methods) work only if the data follow the assumed distribution and this is rarely the case. To produce multiclass decision

boundaries with LDFs, the mean vector $\mathbf{m}_c$ of each class can be calculated and used as an LDF. The VIP of the input with each $\mathbf{m}_c$ and thresholding denotes the class estimate for the input. Criterion functions (error functions) represent a preferable way to select an LDF for each class. One can employ pairwise LDFs (for each LDF, some class i is compared with another class j). These approaches are computationally intensive and not attractive for problems with many classes and they may lead to decision surfaces that have undefined regions (not corresponding to any class).

Thus, standard linear discriminant techniques for multivariate pattern recognition allow us to determine suitable linear discriminants, but these are generally not powerful enough for realistic pattern recognition applications that require nonlinear decision surfaces. In our ACNN, neural net techniques provide refinements to the linear discriminant weight estimates and automatically combine many linear decision boundaries into piecewise linear decision boundaries. We now detail the design and update rules for our ACNN.

A. Selection of the Number of Hidden Layer (Cluster) Neurons

To select the prototypes/exemplars or cluster representatives we use two steps. As our prototypes we desire the N prototypes in the training set whose removal cause the most error in a nearest-neighbor classification. We assume a large training set (N_T samples) for our multiclass problem (so large that simple clustering techniques cannot produce a suitable set of clusters). We first use standard techniques[30] for sample-number reduction to obtain a modest number of prototypes N_R. This reduced nearest-neighbor clustering technique divides the N_T samples into two groups (A and B), where the samples in A classify all N_T samples correctly using a nearest-neighbor technique. Initially, all samples are in group B. The samples in A are used as the prototypes in a nearest-neighbor classifier. Each sample in B is sequentially presented to the nearest-neighbor classifier. If it is incorrectly classified, it is added to A. This procedure is repeated until the samples in group A can correctly classify all N_T samples. (Typically around 5% to 30% of the training samples are still present in N_R and this is still too large ε number of P_2 neurons.)

Thus, we employ a second step to further reduce the number of prototypes (clusters) to an acceptable number N. To achieve this, we remove the first prototype, use the remaining $N_R - 1$ samples in a nearest-neighbor classifier to classify the N_T original samples and calculate the number of misclassifications. We then remove only the second prototype and repeat the above procedure with the remaining $N_R - 1$ samples. This procedure continues until the removal (separately) of each of the N_R prototypes has been tested. If N is prespecified, we keep the N prototypes whose removal would cause the most errors. We can also use the number of errors obtained by removing each prototype to select N (i.e., we select N that results in no more than a given error rate or for which there is a jump in the number of errors produced). We insure that at least one prototype is chosen from each class. Insuring that we keep one prototype per class has not been a problem in our benchmarks (i.e., if the prototypes are ordered by their error rate, we do not find a number of consecutive prototypes in one class before one from another class occurs). In our initial benchmarks, we have not found significant branch points or jumps in the error rates of the ordered samples. There is also no restriction that the same number of prototypes be selected from each class (the data will determine this). Considerable flexibility is possible in how the N prototypes are selected since training will refine the initial choices; therefore, this issue is not of major concern.

This procedure does not account for the fact that, when several samples are not included as prototypes, performance will be worse than when only one of the samples is omitted. However, the purpose of selecting prototypes (or cluster representatives) is only to provide a reasonable or approximate initial selection (the neural net adaptations of these initial choices address the global problem).

We note that use of a nearest-neighbor technique for training is acceptable, but it is not suitable for classification (where on-line real time requirements exist). The combination of our nearest-neighbor prototype selection and ACNN update algorithm will be shown to require fewer iterations than BP. To quantify the significance of this, we now briefly address the number of operations required to select prototypes and relate it to the number of operations required in one BP iteration on all N_T training samples. For each sample, our prototype selection algorithm must calculate the distance to all other points in the training set. For all N_T samples, the claculation of the distances from all points to all points (i.e., the number of distance calculations required for one pass through the N_T training samples) is approximately $0.5N_T^2$ (we precalculate this once and use the 0.5 factor since the calculations are symmetric). In BP, all N_T samples are presented and after each sample we must calculate the activities of all N neurons (N hidden neurons are assumed and the calculation of the activities of the output neurons is ignored), i.e., N_TN calculations are required. The calculation times for the operations in the two cases are equivalent, each is a VIP of dimension equal to that of the feature space used (the calculation times for each operation are exact for the case of layer-one and layer-two neurons). If the additional number of BP iterations required is I, for our algorithm to be computationally efficient, we require

$$0.5N_T^2 < N_TNI. \tag{3}$$

Since $N_T \gg N$ our algorithm may not offer a significant advantage in training time (once N is fixed in BP) unless I is very large.

In obtaining the result in Eq. (3), we assumed that all N_T samples were used in selecting the N prototypes. We have found that we need only use approximately $5N$ randomly selected samples from the full set of N_T in our prototype selection (N is the number of proto-

es or cluster neuron used at P_2 and we have always nd that two to five neurons per class suffice). us, we employ our algorithm using $5N$ samples (not). The inequality to be satisfied is now

$$5(5N)^2 < N_T NI$$
$$5N < 2N_T I. \quad (4)$$

further evaluate this, we assume $N_T \approx 100N$ (this is te typical for distortion-invariant problems to adeately represent all distortions). We then find

$$5 < 200I \quad (5)$$

ich is independent of N. This inequality is always isfied. As we shall see, BP has always required at st on the order of $I = 100$ more iterations of the full ining set than has our ACNN algorithm. In this e

$$5 < 2 \times 10^4. \quad (6)$$

l the computational time savings of our algorithm is ite significant.

Thus, to summarize, in the two steps of our proto-e selection algorithm we use $5N$ random samples m the full N_T set. We select the number of hidden irons N to be two to five times the number of classes pending on the difficulty of the problem). Sec. IV ails these choices for two examples.

Initial P_1–P_2 Weights

We now address how we select the initial P_1–P_2 put-to-hidden layer) weights. We denote the ight between input neuron j and hidden neuron i by . We denote the vector position of prototype i in r D-dimensional feature space by $\mathbf{p}_i$ (i.e., this is the ture vector for prototype i) and element j of it by p_{ij}. e can now describe the input weights from P_1 to P_2 as

$$w_{ij} = \begin{cases} p_{ij} \text{ for } j = 1, \ldots, D \\ -(1/2)\sum_{i=1}^{D} p_{ij}^2 \text{ for } j = D+1. \end{cases} \quad (7)$$

e first D (out of $D + 1$) elements of each weight tor from P_1 to layer-two neuron i are thus the fea-e vector $\mathbf{p}_i$ associated with that prototype. The last + 1) input neuron activity is always 1 and its weight hidden layer neuron i is associated with its LDF eshold. We choose these initial weights since they ure that the classifier initially implements a near-neighbor classifier based on the prototypes, as we v detail.

Each hidden neuron i has connections from all $D + 1$ ut neurons and thus has a weight vector $\mathbf{w}_i$ associatwith it. For an input $\mathbf{x}_a$, the input to neuron i in r two is

$$\mathbf{w}_i^t \mathbf{x}_a - 0.5\, p_{ij}^2, \quad (8)$$

re the first term is the contribution to the VIP from first D weights and the last term is the contribution to the additional $D + 1$ input neuron. We rewrite (8) as

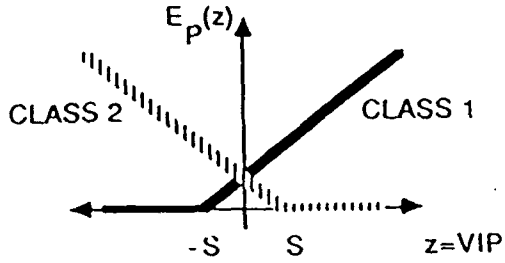


Fig. 2. Perceptron criterion function: S denotes the safety margin and the solid and dashed curves correspond to classes 1 and 2, respectively.

$$\mathbf{w}_i^t \mathbf{x}_a = (0.5)2\mathbf{p}_i^t \mathbf{x}_a - 0.5\mathbf{p}_i^t \mathbf{p}_i + 0.5\mathbf{x}_a^t \mathbf{x}_a - 0.5\mathbf{x}_a^t \mathbf{x}_a$$
$$= 0.5[\mathbf{x}_a^t \mathbf{x}_a - (\mathbf{p}_i^t \mathbf{p}_i - 2\mathbf{p}_i^t \mathbf{x}_a + \mathbf{x}_a^t \mathbf{x}_a)]$$
$$= 0.5(\|\mathbf{x}_a\|^2 - \|\mathbf{p}_i - \mathbf{x}_a\|^2). \quad (9)$$

From Eq. (9) we see that the VIP is related to the Euclidean distance (denoted by $\| \ \|$) between the input $\mathbf{x}_a$ and the prototype $\mathbf{p}_i$ associated with hidden neuron i. The choice of weights in Eq. (7) achieves nearest-neighbor classification since it ensures, from Eq. (9), that the hidden neuron closest to $\mathbf{x}_a$ will have the largest input (since the second term in Eq. (9) is then smallest) and will be most active.

C. Training (Weight Update) Algorithm

We now detail how we update the initial P_1–P_2 weights to achieve improved piecewise linear decision surfaces. We input each of the full N_T set of training vectors $\mathbf{x}_a$. For each $\mathbf{x}_a$ we calculate the most active hidden neuron $i(c)$ in the proper class c and the most active one $i(\bar{c})$ in any other class ($\bar{c}$). We denote the weight vectors for these two layer-two neurons by $\mathbf{w}_{i(c)}$ and $\mathbf{w}_{i(\bar{c})}$ and their VIPs with the input by $\mathbf{w}_{i(c)}^t \mathbf{x}_a$ and $\mathbf{w}_{i(\bar{c})}^t \mathbf{x}_a$. The perceptron error function (criterion function) E_P used is shown in Fig. 2. The solid (dashed) curves correspond to the true (false) classes 1 and 2 cases. The offset S is a safety margin that forces training set vectors which are classified correctly by a small amount ($<S$) to also contribute to the criterion function. As discussed elsewhere,[26] we chose $S = 0.05$ (all features were normalized between 0 and 1). The use of S forces the classifier to try to classify all training samples correctly by at least an amount S, improving test set performance (and thus generalization).

For each training sample in N_T, we add an error (penalty) to E_P. The error added is

$$E = 0 \text{ if } \mathbf{w}_{i(c)}^t \mathbf{x}_a > \mathbf{w}_{i(\bar{c})}^t \mathbf{x}_a + S$$
$$S + (\mathbf{w}_{i(\bar{c})} - \mathbf{w}_{i(c)})^t \mathbf{x}_a \text{ otherwise}, \quad (10)$$

where the $E = 0$ case corresponds to the situation when the proper layer-two neuron is most active (by an amount S above the most active false neuron) and where the other case corresponds to the situation when the false class VIP is larger than the true class VIP, or within S of it.

After all N_T training samples have been run through the system, we accumulate all of these errors or ener-

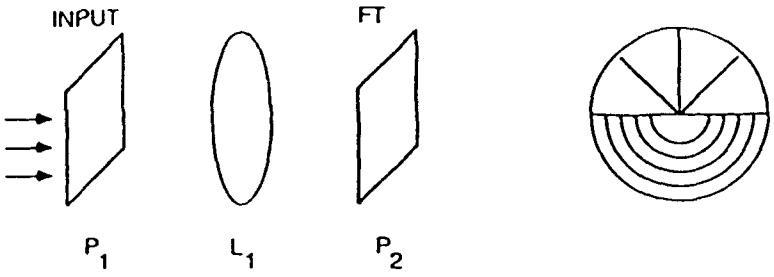


Fig. 3. Input P_1 neuron representation space (wedge sampled Fourier transform): (a) architecture; (b) P_2 sampling.

gies (all are positive or zero). We also accumulate the gradients $\nabla_{w_i}E$. From Eq. (10), by taking the derivative with respect to w_i, we see that $\nabla_{w_i}E$ is zero for all i when an input is classified correctly by more than S; otherwise, it equals either $\mathbf{x}_a$ (if input a should be classified into the same class as cluster-neuron i) or $-\mathbf{x}_a$ (if a is incorrectly classified by cluster neuron i). Thus, the sum of all the contributions to $\nabla_{w_i}E$ equals the sum of the $\pm\mathbf{x}_a$ for samples erroneously classified (or correctly classified but with a margin less than S) in layer-two clusters. We then use $\nabla_{w_i}E$ to adapt the weights $\mathbf{w}$ by the conjugate-gradient algorithm. We then repeat presentation of the training set (a new iteration), calculate the new errors E and $\nabla_{w_i}E$ and update the weights accordingly. This procedure repeats until satisfactory performance on the test set is obtained.

We considered other LDFs (Ho-Kashyap, Fisher,

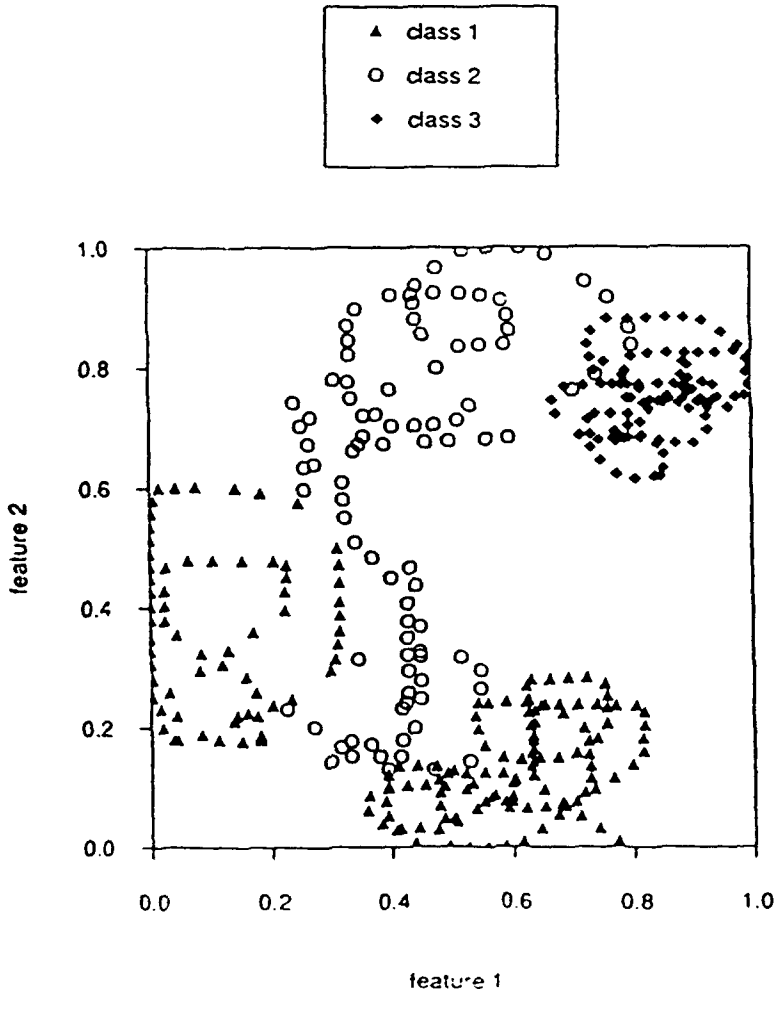


Fig. 4. Three-class, two-feature artificial database example (benchmark 1).

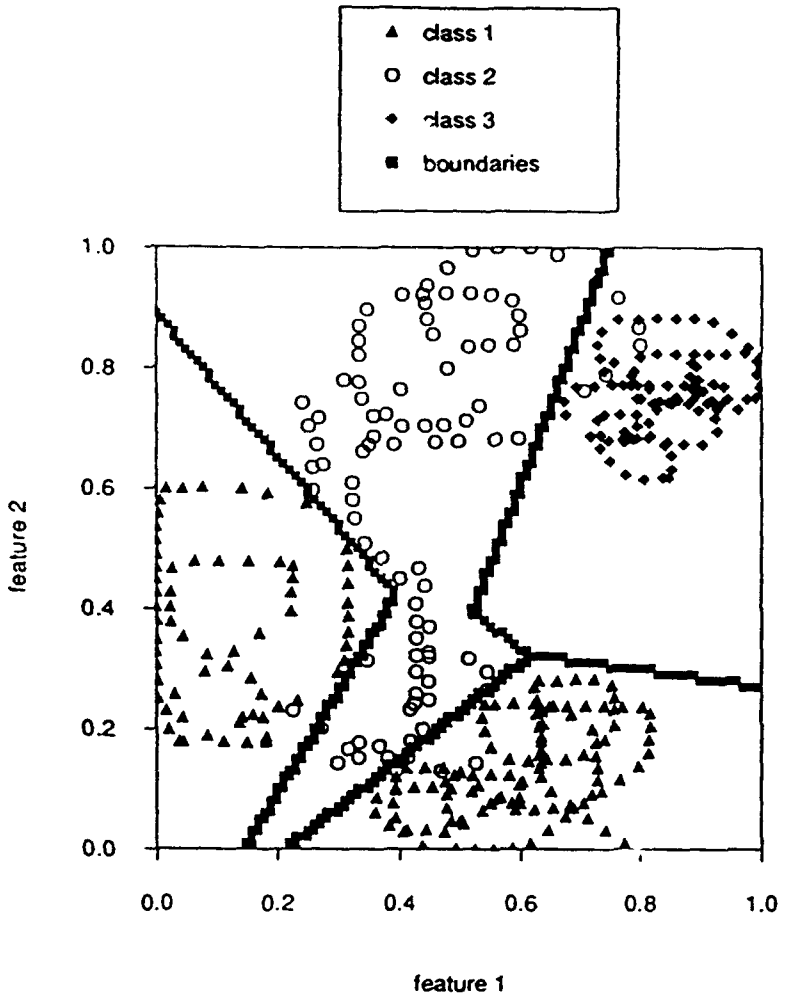


Fig. 5. Nonlinear decision boundaries produced for the artificial database.

Fukanaga-Koontz, etc.). However, these LDFs require more calculation than does our current algorithm to update the weights. Thus, for computational reasons, our present choice (perceptron criterion) is preferable.

D. Input P_1 Neuron Representation Space

In our distortion-invariant multiclass pattern recognition applications, we use a wedge-sampled magnitude Fourier transform feature space,[31] since this feature space can easily be produced optically. Figure 3(a) shows the standard architecture that produces the Fourier transform at P_2 of the P_1 input 2-D image data. Figure 3(b) shows the standard wedge-ring detector used at P_2. The wedge features provide scale invariance and the ring features provide in-plane rotation invariance. Our distortion-invariant data will involve different aspect views of several objects (and not in-plane distortions). Thus, we chose the wedge features (this provides scale invariance, although we do not include scale distortions in our test data). We obtain aspect-view invariance by training on various aspect-distorted object views.

IV. Test Results

We consider two databases: an artificial set of data[15,26] (to demonstrate the nonlinear surfaces produced using only two features) and a set of three aircraft with various azimuth and elevation (3-D) distortions present. We refer to these as benchmarks 1 and 2.

A. Benchmark 1 Results (Artificial Data)

An artificial set of 383 samples in three classes (181 in class 1, 97 in class 2 and 105 in class 3) with two features was generated with samples as shown in Fig. 4. This problem definitely requires a nonlinear decision boundary and the results can be shown in the 2-D feature space. This is the purpose of this example, since no separate test data exist. The neural net used contained three input neurons (two for the features plus one for the threshold), six hidden neurons (two per class) and three output neurons (one per class). All N_T samples were used to select the prototypes. The first reduced nearest-neighbor clustering produced thirty-one prototypes (8.1% of the total N_T) that

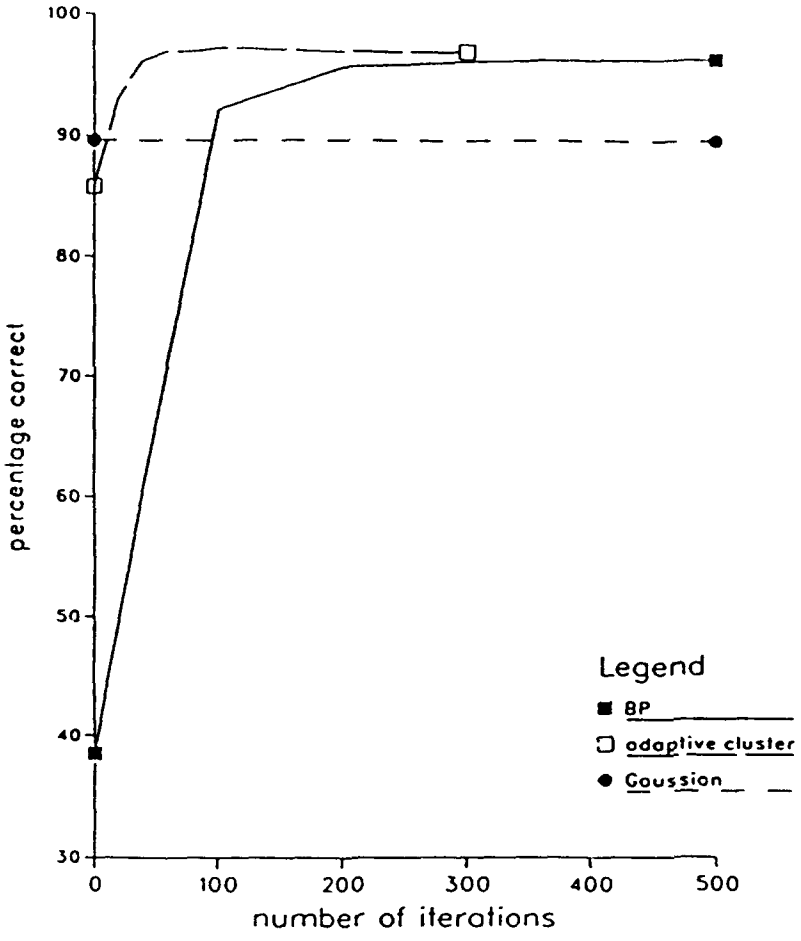


Fig. 6. Comparative data on speed of convergence for benchmark-1 data.

gave an error rate P_e = 0% for all samples. The six prototypes whose removal gave the most error were then selected in stage two.

After eighty iterations of the full training set, the classification rate (defined as the percentage of test samples correctly classified) was constant at 97.1% with our ACNN algorithm. After 300 iterations the BP classification rate was constant at approximately the same value (96.3%). (This result is the average obtained over ten runs with different random initial weight sets.) The final input layer weights to the six hidden layer neurons correspond to six straight lines (LDFs) in the feature space. For BP these six lines would define the decision surface. In the ACNN this is not the case (because of the winner-takes-all action at P_2). The decision-surface lines were determined by successively providing all of the possible feature vectors on a grid of $x_1 - x_2$ values (for both x_1 and x_2 in the interval [0,1]) to the classifier and for each feature vector determining the class into which it is classified by the neural net. The decision boundaries indicate where a transition in classification occurred. The boundaries thus obtained are shown in Fig. 5. They produce four separate regions of feature space (two correspond to the same class and the others correspond to the other two classes).

From inspection of Fig. 4 one would estimate that a piecewise linear decision surface with at least five straight-line sections would be needed to separate the data adequately and that about ten errors might be expected. Thus, at least five hidden neurons are expected to be needed. In Fig. 4 we see that, with six hidden neurons, approximately ten classification errors are made, producing the error rate of 97.1%.

Figure 6 compares the classification rate for the two neural nets and for a multivariate Gaussian classifier. Both neural nets give comparable classification rates (97.1% and 96.3%) after convergence, whereas the Gaussian classifier's performance is worse (89.5%) and by definition does not vary with the number of iterations of the training set. The speed of learning of the ACNN is much faster (convergence in 80 iterations) than for BP (approximate convergence in 300 iterations). From Eq. (4) this represents approximately an additional $N_T NI = (383)(6)(220) = 505560$ VIP calculations required with BP. The prototype selection steps in our ACNN algorithm required approximately $0.5N_T^2 = (0.5)(383)^2 \approx 73350$ VIPs and thus the total number of calculations and hence training time for our ACNN is considerably less than the learning time for BP. We reran the prototype selection portion of our ACNN algorithm using only $5N = 30$ samples random-

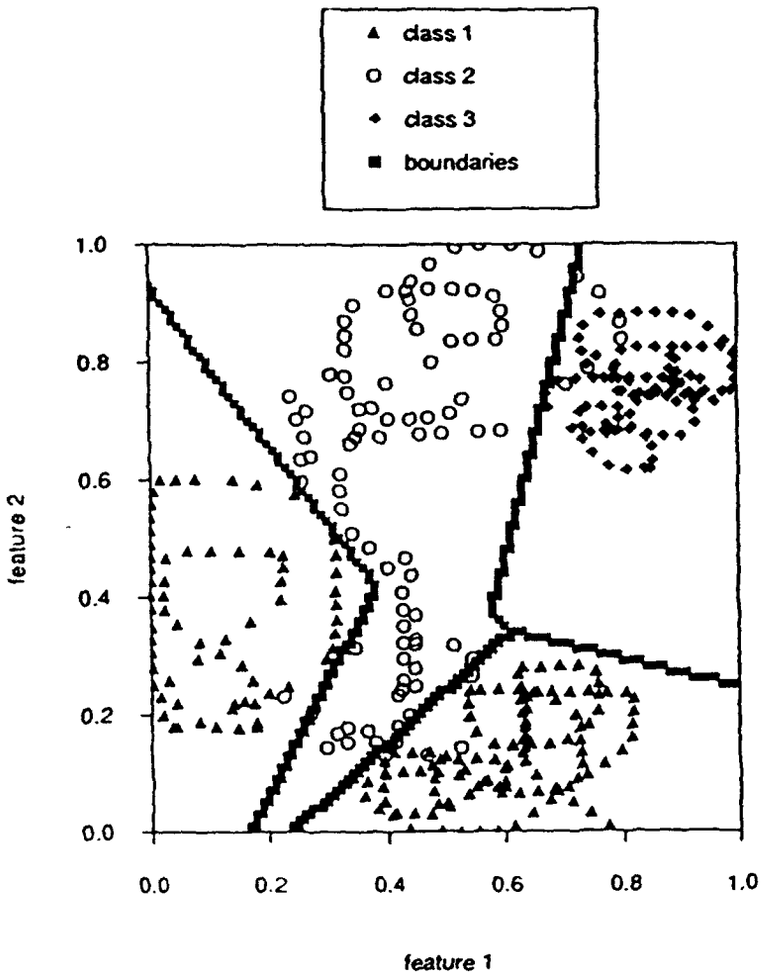


Fig. 7. Nonlinear decision boundaries produced for the artificial database when prototypes are selected from reduced training set.

ly selected from the 383, insuring that we obtain at least one prototype per class. The decision boundaries produced are shown in Fig. 7. As can be seen, the decision boundaries are virtually identical; the resulting error rates differ by only 0.2% (96.9% classification was obtained after 100 iterations). This was now achieved with only $0.5(30)^2 = 450$ VIPs for prototype selection.

This data set, therefore, indicates that similar performances can be obtained with BP and ACNN, with ACNN training appreciably faster than BP. We have also seen that the time for prototype selection with ACNN can be made negligible by using a reduced number of learning samples, without affecting performance adversely.

B. Benchmark 2 Results (3-D Distorted Aircraft Data)

As our second data set, we used synthetic distorted aircraft imagery and our wedge-sampled Fourier feature space. The imagery used were three aircraft (F-4, F-104, and DC-10) binarized to 128 × 128 pixels with each aircraft occupying about the central 100 × 64 pixels. As our training set, we used 630 images of each aircraft (a total of $N_T = 1890$ training set samples). The images were different azimuth views (with the aircraft viewed from different angles left to right) and elevation views (with the aircraft viewed from different angles above or below its center line). The range of azimuth angles used covered −85° to +85° and the elevation angle was varied from 0° to 90° with 5° increments in each angle (the same image results if negative elevation angles are used). The input neuron representation space was a thirty-two element feature space (the thirty-two wedge magnitude Fourier samples). The test set used consisted of 578 orientations of each aircraft not present in the training set (these were views at internal angles about 2.5° in each direction from those in the training set). Figure 8 shows three distorted versions of each aircraft. The left image is the top-down view with 0° variation in elevation and azimuth. The central image shows a view from an azimuth angle of 45° to the left. The right image for each object shows an image with elevation angle of 45°.

The three-layer ACNN used contained thirty-three input neurons, nine hidden neurons and three output neurons (one per class).

Figure 9 compares the speed (number of iterations of the full training set) and classification performance for

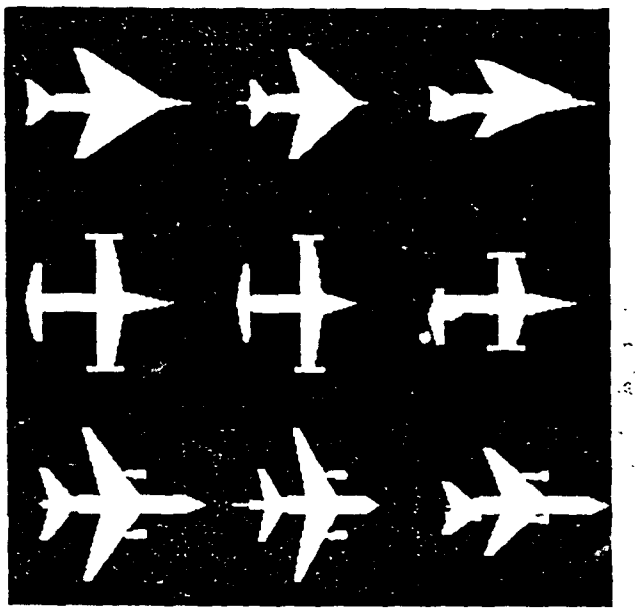


Fig. 8. Representative images for the three-class 3-D distortion example (benchmark 2).

the two neural nets and the Gaussian classifier. Both neural nets yield the same classification rate (98.6%) compared to only 89% for the Gaussian classifier. BP converges in 350 iterations and our ACNN in fewer (180) iterations. As with the 2-D data set, a reduced data set for prototype selection can be employed successfully. It was found that with $5N = 45$ samples used for prototype selection, 98.6% classification performance was obtained after 180 iterations. With this reduced number of samples, the time for prototype selection is negligible compared with the time for a single iteration, so that the relative training times are again determined by the number of iterations required for each method. Thus, ACNN requires approximately 50% of the training time of BP.

V. Optical and Optical/Electronic Realization

Many choices are possible for the role of optics in the learning and classification stages of our ACNN. These are now discussed. The feature space (wedge-sampled magnitude Fourier transform) should be optically calculated (even in learning) since this feature space is easily produced optically[32,33] and since we will

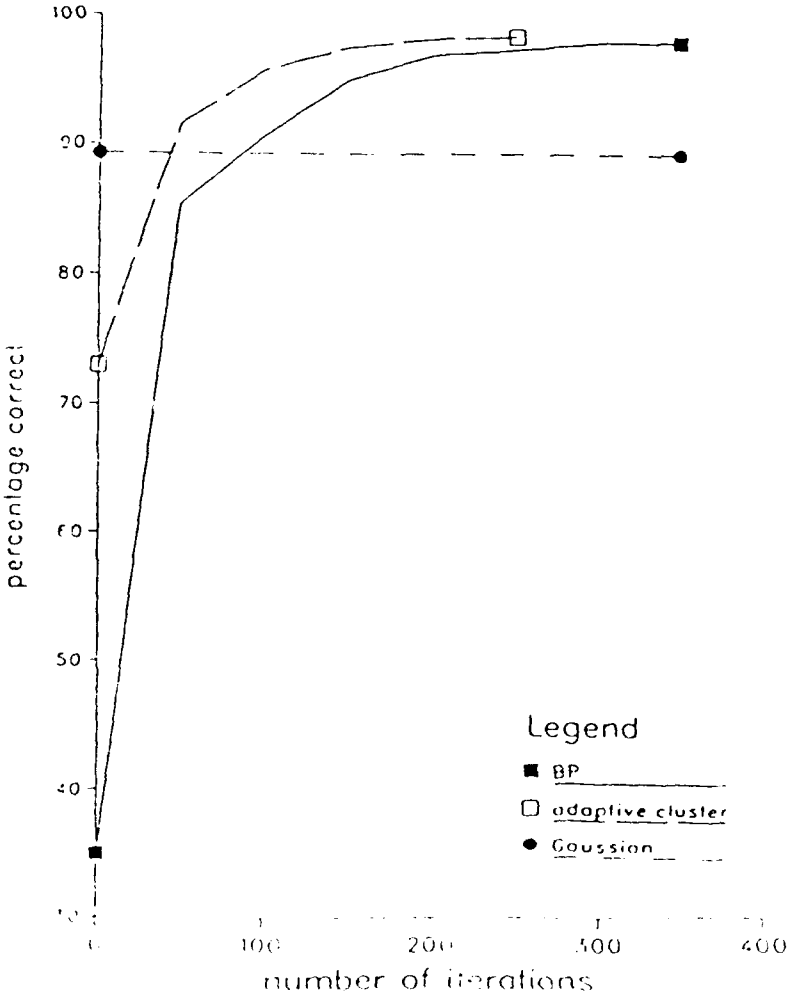


Fig. 9. Comparative data on speed of convergence for benchmark-2 data.

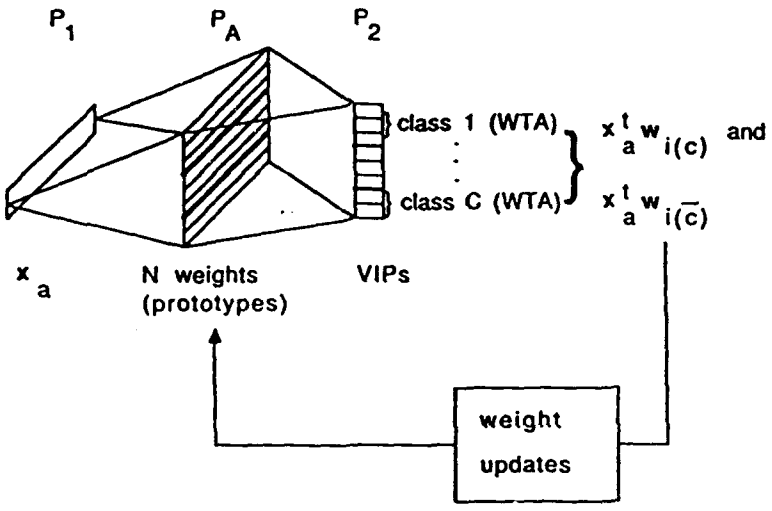


Fig. 10. Possible optical architecture for adaptive learning.

use the optically produced feature space in our on-line classification. The two steps of prototype selection are best performed electronically, since they are off-line operations and require manipulation of stored data and control operations most compatible with digital electronics. The distance calculations required in the nearest-neighbor calculations can be performed on an optical VIP architecture (we now discuss this and the use of optics in the learning stage).

Once the initial P_1–P_2 weights have been chosen, the learning stage can be implemented in optics or electronics. Figure 10 shows one such architecture. The input sample $\mathbf{x}_a$ is entered at P_1 (on LEDs, laser diodes or a 1-D spatial light modulator (SLM)). It is imaged onto the initial set of N weight vectors (for the N prototype hidden layer neurons) which are arranged on rows at P_A (with the first two to five rows corresponding to the prototypes for class 1, the next two to five rows being the prototypes for class 2, etc.). Thus, the rows at P_A are the initial weights as given in Eq. (7). The VIPs of $\mathbf{x}_a$ and all of the $\mathbf{w}_i$ weight vectors at P_A are formed on a linear detector array at P_2. The P_A rows and P_2 elements are separated into C groups (the C classes). The maximum VIP element in each class is determined (simple comparator logic is sufficient since the number of prototypes per class is small). This provides us with $\mathbf{w}_{i(c)}^t\mathbf{x}_a$ and $\mathbf{w}_{i(\bar{c})}^t\mathbf{x}_a$ in Eq. (10). Bipolar values for $\mathbf{w}_i$ should be handled by spatial multiplexing at P_A and subtraction of adjacent P_2 outputs. Alternatively, the P_A data can be placed on a bias (but this increases dynamic-range requirements). The weights must be updated after each iteration of the training set. If P_A is a microchannel spatial light modulator[34] (or similar device) that can record positive and negative data (with a bias on the device), we can update the weights by adding and/or subtracting the appropriate values for each weight. These updates to the weights at P_A are various combinations of the training vectors $\mathbf{x}_a$. These could be calculated in electronics, entered sequentially at P_1 and (with a mechanism to activate only selected rows at P_A) we could update P_A as required. Alternatively, we could repeat each $\mathbf{x}_a$ at P_1 and vary the input illumination and the P_A row accessed and hence control the amount of each $\mathbf{x}_a$ added to or subtracted from each weight vector at P_A. The digital control required, the complexity of the system (a modulated light source to control the amount of each $\mathbf{x}_a$ used, access to only one row of P_A at a time), the need for N accesses of P_A for each of the N_T vectors $\mathbf{x}_a$, and the P_A SLM requirements make the electronic calculation of the updated weight vectors and the electronic off-line implementation of the learning stage preferable (at present). As P_A SLM technology matures, it would probably be realistic to calculate all VIPs optically, determine the new weights electronically, and reload these directly into P_A after each iteration of the training set. However, at present, we assume that all learning is electronic (since it is off-line).

Once learning has been completed, the P_1–P_2 weights are fixed and the input-to-hidden layer neurons and weights (the P_1–P_2 neuron system) can be implemented on an optical VIP system (such as P_1–P_2 of Fig. 10) with a fixed mask at P_A. The number of P_1 neurons is modest (the input neuron representation is a compact feature space), and the number of P_2 neurons is also small (typically less than five times the number of classes). Our ACNN requires a winner-takes-all (WTA) maximum selection of the most active P_2 neuron. This can be implemented with a WTA neural network or in standard comparison techniques. Since the number of P_2 neurons (N) is small, standard electronic WTA techniques are preferable (we quantify this below). Since the P_2–P_3 hidden-to-output neuron weights are fixed and are all unity or zero, the P_2–P_3 weights simply perform a mapping and can easily be implemented in electronics. Thus, we implement the input-to-hidden layer neuron weights and calculations optically and the hidden layer neuron maximum selection (WTA) and the hidden-to-output neuron mapping in electronics. Figure 11 summarizes the learning and classification stages in block diagram form with

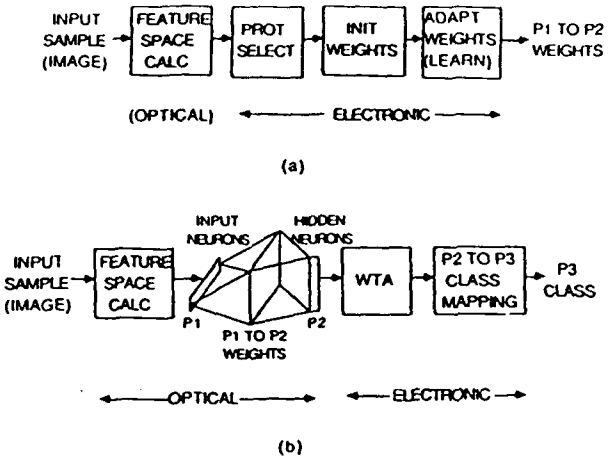


Fig. 11. Block diagram for adaptive-clustering neural net using (a) electronics for learning (training) and (b) optics for classification (on-line).

attention to which operations are performed in optics and which in electronics.

The two WTA electronic techniques possible (in classification) are to use an operational amplifier peak detector to scan all N outputs at P_2 or to employ a parallel digital technique. In the digital technique, the N outputs are A/D converted, each pair of P_2 outputs (1 and 2, 3 and 4 etc.) are pairwise compared and the maximum of each pair is obtained. Pairwise comparisons of the $N/2$ outputs are then performed and the procedure is continued for $\log_2 N$ levels until the maximum is obtained. For 100 input and hidden neurons, one matrix–vector multiplication (required to update the P_2 neuron activities) requires about 10,000 additions and 10,000 multiplications; whereas, maximum selection requires only about 100 comparisons. Thus, the maximum selection is typically negligible computationally compared with the neuron update stage, and can be implemented in serial electronic hardware without sacrificing the speed of the system. We, thus, implement the WTA operation in electronics using comparators rather than with a neural net. The specific electronic WTA technique chosen depends on the accuracy and speed required. Since these operations are required once for each test input in classification, the WTA time required is set by the rate at which new input image data occurs and the rate at which its features can be calculated.

VI. Summary, Conclusions and Discussion

A new three-layer adaptive-clustering neural net (ACNN) has been described. It provides for a new procedure to select the number of hidden layer neurons (we use several neurons per class, each being a prototype or cluster representive of a particular class) and provides initial (non-random) input-to-hidden layer neuron weights. These initial weights are selected using standard pattern recognition clustering techniques. They are then updated during learning using a new neural net adaptive supervised learning algorithm. This results in a new neural net that combines standard pattern recognition and neural net techniques to produce piecewise linear decision surfaces from the linear discriminant functions. The input neurons are analog and of low dimensionality (a feature space with inherent distortion invariances). Quantitative data show that the learning time and number of calculations required in our new ACNN is significantly faster (by a factor of 2 to 4) than the more well-studied BP neural net. We also found that the use of a conjugate-gradient (rather than gradient descent) update algorithm significantly speeds up BP.

BP and the ACNN will usually not result in similar weights since BP uses neurons for other operations besides clustering, because BP has no WTA competition in its hidden layer as in the ACNN and because the hidden-to-output weights are different in BP and only perform mapping in the ACNN. However, the decision boundaries that result are usually very similar (with the ACNN decision boundaries generally being a piecewise linear approximation to the more curved ones in BP). Thus, the two classifiers employ different means to similar ends, with the ACNN providing faster training without the need to select many empirical parameters. Since only one hidden neuron in ACNN is dominant, piecewise linear surfaces result and more hidden neurons may be needed. Our intent is not to compare BP and our ACNN, rather we note the attractive properties of our new neural net. Besides providing a new way to select the hidden neurons, our neural net algorithm has only one *ad hoc* parameter to be empirically selected (the number of hidden neurons). Changes in ACNN weights during training provide information on the data that can be of use in better understanding results and in extending results to other cases (other neural nets do not have this property). For example, in sequential gradient descent updating algorithms (the delta rule) different results occur depending on the order in which the training data are presented and depending on the random initial weights (by comparison, the ACNN provides consistent results).

This work was supported by a contract (DAAH01-89-C-04180) from the Defense Advanced Research Project Agency, monitored by the U.S. Army Missile Command.

References

1. *Technical Digest, International Conference on Neural Networks, IEEE* (IEEE Press, San Diego, 1988).
2. *Technical Digest, International Joint Conference on Neural Networks, IEEE International Neural Network Society* (IEEE Press, Washington, DC, 1989).
3. *Advances in Neural Information Processing Systems* (M. Kaufmann, Boulder, CO, 1988).
4. K. Wagner and D. Psaltis, "Multilayer optical learning networks," Applied Optics 26, 5061–5076 (1987).
5. H. Yoshinaga, K-I. Kitayama and T. Hara, "All-Optical Error-Signal Generation for Backpropagation Learning in Optical Multilayer Networks," Opt. Lett. 14, 202–204 (1989).

6. D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing* (MIT press, Cambridge 1986), chap. 8, 318–362.
7. N. Farhat, "Optoelectronic Analogs of Self-Programming Neural Nets: Architecture and Methodologies for Implementing Fast Stochastic Learning by Simulated Annealing," Appl. Opt. 26, 5093–5103 (1987).
8. D. Z. Anderson and D. M. Lininger, "Dynamic Optical Interconnects: Volume Holograms as Two-Port Operators," Appl. Opt. 26, 5031–5038 (1987).
9. A. D. Fisher, W. L. Lippincott, and J. N. Lee, "Optical Implementations of Associative Networks with Versatile Adaptive Learning Capabilities," Appl. Opt. 26, 5039–5054 (1987).
10. D. Psaltis and N. Farhat, "Optical Information Processing Based on an Associative-Memory Model of Neural Nets with Thresholding and Feedback," Opt. Lett. 10, 98–100 (1985).
11. B. H. Soffer, G. J. Dunning, Y. Owechko, and E. Marom, "Associative Holographic Memory with Feedback Using Phase-Conjugate Mirrors," Opt. Lett. 11, 118–20 (1986).
12. L-S. Lee, H. M. Stoll and M. C. Tackitt, "Continuous-Time Optical Neural Network Associative Memory," Opt. Lett. 14, 162–164 (1989).
13. K. Hsu and D. Psaltis, "Invariance and Discrimination Properties of the Optical Associative Loop," in *Technical Digest, International Conference on Neural Networks* (IEEE, San Diego, 1988), pp. II-395–II-402.
14. E. C. Botha, D. Casasent and E. Barnard, "Optical Neural Networks for Image Analysis: Imaging Spectroscopy and Production Systems," in *Technical Digest, International Conference on Neural Networks* (IEEE, San Diego, 1988), pp. I-541–I-546.
15. E. Barnard and D. Casasent, "Image Processing for Image Understanding with Neural Nets," in *Technical Digest, International Joint Conference on Neural Networks* (IEEE, Washington, DC, 1989), pp. I-111–I-116.
16. J. J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," Proc. Nat. Acad. Sci. USA 79, 2554-2558 (1982).
17. J. J. Hopfield and D. W. Tank, "Computing with Neural Circuits: A Model," Science 233, 625–633 (1986).
18. G. E. Hinton and J. A. Anderson, Eds., *Parallel Models of Associative Memory* (Laurence Erlbaum Assoc., Hillsdale, NJ, 1981).
19. B. Montgomery and B. V. K. Vijaya Kumar, "An Evaluation of the Use of the Hopfield Neural Network Model as a Nearest-Neighbor Algorithm," Appl. Opt. 25, 3759–3766 (1986).
20. T. Kohonen, G. Barna and R. Chrisley, "Statistical Pattern Recognition with Neural Nets: Benchmarking Studies," in *Technical Digest, International Conference on Neural Networks* (IEEE, San Diego, 1988), pp. I-61–I-68.
21. B. Kosko, "Bidirectional Associative Memories," IEEE Trans. Syst., Man, Cybern. SMC-18, 49–60 (1988).
22. E. B. Baum, "On the Capabilities of Multilayer Perceptrons," J. Complexity 4, 193–215 (1988).
23. G. Mirchandani and W. Cao, "On Hidden Nodes for Neural Nets," IEEE Trans. Circuits Syst. CAS-36, 661–664 (1989).
24. B. Irie and S. Miyake, "Capabilities of Three-Layered Perceptrons," in *Technical Digest, International Conference on Neural Networks* (IEEE, San Diego, 1988), pp. I-641–I-648.
25. M. Stinchcombe and H. White, "Universal Approximation Using Feedforward Networks with Non-Sigmoid Hidden Layer Activation Functions," in *Technical Digest, International Conference on Neural Networks* (IEEE, Washington, DC, 1989), pp. I-613-I-618.
26. E. Barnard and D. Casasent, "A Comparison Between Criterion Functions for Linear Classifiers, with an Application to Neural Nets," IEEE Trans. Syst., Man, Cybern. SMC-19, 000–000 (1989).
27. M. J. D. Powell, "Restart Procedures for the Conjugate Gradient Method," Math. Program. 12, 241–254 (1977).
28. A. Lapedes and R. Farber, "How Neural Nets Work," *Neural Information Processing Systems*, D. Z. Anderson, Ed. (AIP, Denver, Co, 1988), pp. 442–456.
29. E. Barnard, E. Botha and D. Casasent, "Neural Nets as Piecewise Linear Classifiers: New Algorithms," Neural Networks 1, Supplement 73 (1988).
30. T. M. Cover and P. E. Hart, "Nearest Neighbor Pattern Classification," IEEE Trans. Inf. Theory IT-13, 21–27 (1967).
31. G. G. Lendaris and G. L. Stanley, "Diffraction-Pattern Sampling for Automatic Pattern Recognition," Proc. IEEE 58, 198–205 (1979).
32. H. Kasdan and D. Mead, "Out of the Laboratory and into the Factory—Optical Computing Comes of Age," in *Proceedings Electro Optical System Design* (1975), pp. 248–258.
33. D. F. Clark and D. P. Casasent, "Practical Optical Fourier Analysis for High Speed Inspection," Opt. Eng. 27, 365–371 (1988).
34. C. Warde and J. Thackara, "Operating Modes of the Microchannel Spatial Light Modulator," Opt. Eng. 22, 695–699 (1983).

Ho-Kashyap optical associative processors

Brian Telfer and David P. Casasent

A Ho-Kashyap (H-K) associative processor (AP) is shown to have a larger storage capacity than the pseudoinverse and correlation APs and to accurately store linearly dependent key vectors. Prior APs have not demonstrated good performance on linearly dependent key vectors. The AP is attractive for optical implementation. A new robust H-K AP is proposed to improve noise performance. These results are demonstrated both theoretically and by Monte Carlo simulation. The H-K AP is also shown to outperform the pseudoinverse AP in an aircraft recognition case study. A technique is developed to indicate the least reliable output vector elements and a new AP error correcting synthesis technique is advanced.

I. Introduction

The storage capacity,[1-4] noise performance[5-7] and key vector requirements[8] of associative processors (APs) are of major concern. This paper addresses these issues using a new AP. It is important to distinguish between general memory and pattern recognition applications. In a general memory application, APs store arbitrary data and it is fair to assume that the keys (input vectors) and recollections (output vectors) are drawn from random distributions (these APs are tested with Monte Carlo methods). We define the storage capacity of an AP to be the number of key/recollection vector pairs that can be nearly perfectly (99–100%) stored in a general memory application. In pattern recognition problems, an AP has many key vectors (e.g., distorted inputs) associated with the same recollection vector (a class label) and must generally operate on shifted and distorted input patterns. A large number of keys are stored to represent the distortions of the different classes. In pattern recognition, recall accuracy for a specific use is more important than storage capacity.

We consider APs with bipolar binary recollection vectors. This case commonly occurs in pattern recognition, where the recollection vectors are class labels. Our key vectors have analog values taken from arbitrary data (for the general memory) or a feature space (for the pattern recognition application). We consider heteroassociative processors (HAPs), in which the keys and recollections differ, rather than autoassociative processors (AAPs), in which the keys are identical to their recollections.

One popular AP, the Hopfield memory, has been shown empirically[2] to have a capacity of $M \approx 0.15N$, where M is the number of keys and N is their dimension. Theoretically, an asymptotic ($N \to \infty$) capacity of $M = N/(4 \log_2 N)$ has been shown for the Hopfield memory.[3] Because of its very low capacity, we do not further consider this or similar correlation APs[5] (where the memory matrix is calculated by summing the vector outer product of each key and its recollection). The pseudoinverse AP[5] (where the memory matrix is calculated from the pseudoinverse of the key matrix) is preferable because it perfectly stores key/recollection vector pairs as long as the keys are linearly independent. This allows up to $M = N$ vector pairs to be perfectly stored. The pseudoinverse AP also has good recall accuracy when $M > N$.[1] In this paper, we discuss how an AP computed by the Ho-Kashyap (H-K) algorithm[9] has better recall accuracy and a larger capacity than the pseudoinverse memory.[4] For general memory applications, we show[4,10] that the maximum storage capacity of the H-K AP is $M = 2N$, and that it can perfectly store keys that are linearly dependent. We also modify the H-K AP to improve its noise performance. A modified version of the algorithm (an error correcting H-K AP algorithm) that allows a low accuracy processor to be used is also advanced. This is of particular concern when an optical processor is employed.

Other AP work[11-16] used the H-K algorithm for computing APs, but limited the number of key vectors to be substantially less than their dimensionality. [An overdetermined problem was created by adding key/recollection vector constraints to map unit vectors

The authors are with Carnegie Mellon University, Department of Electrical & Computer Engineering, Center for Excellence in Optical Data Processing, Pittsburgh, Pennsylvania 15213.

Received 18 May 1989.
0003-6935/90/081191-12$02.00/0.

(with a single 1) to all-zero vectors, in addition to storing the key vectors.] A major emphasis of this paper is that the advantage of the H-K AP over the pseudoinverse AP occurs when the number of key vectors exceeds their dimensionality and that the H-K AP can handle linearly dependent key vectors (i.e. $M > N$). Other H-K AP work also used output thresholding and feedback in recall mode. We do not consider this for high capacity APs ($M > N$). To use feedback in HAP recall requires a bidirectional associative memory (BAM).[17] To use the H-K algorithm to synthesize a BAM requires that two separate forward (key → recollection) and reverse (recollection → key) mappings be computed. This significantly increases the complexity of the processor and the amount of storage required, and hence makes the bidirectional processor less desirable for optical implementation. Also, the BAM capacity is limited by the minimum of N and K, where N and K are the key and recollection vector dimensions.[17] When K is small (which is to be expected in an HAP when the recollections are used for decisions or class labelling), the BAM's capacity (where the BAM is constructed using the H-K algorithm) will cause the memory's capacity to be less than that for the unidirectional processor we consider.

Another approach to associative storage is the direct storage nearest-neighbor (DSNN) AP.[18] For bipolar binary keys, the memory matrix simply contains the key vectors as its rows. In recall, the output vector resulting from multiplying the memory matrix by the input vector has elements that are the vector inner products of the input with each key. The largest output element indicates which key has the smallest Hamming distance to the input. The corresponding recollection vector can then be selected as the final output. The Hamming Net[19] operates on the same principles. The DSNN AP can also be extended to analog keys. The AP then finds the key with the smallest Euclidean distance to the input. The DSNN AP has several attractive properties. It is trivial to synthesize and to update, and it is guaranteed to output the recollection whose the key is closest to the input.

The other APs that we have mentioned (correlation, pseudoinverse, H-K) are more difficult to update (although the correlation AP is still relatively simple to update). They are also not guaranteed to output the recollection whose key is closest to the input, although they do so for low input noise levels. We believe that the main advantage of these three APs over the DSNN AP is that their memory matrices can be smaller.[8] The DSNN AP memory matrix has MN elements, while the other AP memory matrices each have KN elements, where K is the recollection vector dimension. Thus, the other APs have fewer matrix elements than the DSNN AP when $K < M$. This condition is true when the recollections are class labels and have a low dimension. These low dimensional labels from an AP can be used to read out high dimensional recollection vectors from an addressable memory. In addition, the H-K AP can store $M > N$ key vectors, and in this case $K < M$ even if $K = N$. For optical implementations, where space bandwidth product is a major concern, the difference in memory matrix size is important. The longer updating times for the pseudoinverse and H-K APs are not a major concern for applications utilizing gated learning,[20] where most time is spent in recall mode, and learning is only initiated after a significant event has occurred.

In Sec. II, we review the pseudoinverse AP and establish our notation. Section III advances our H-K algorithms. Optical implementation of these processors is considered in Sec. IV. Section V gives theoretical and simulation results for the general memory application. A case study of distortion invariant aircraft recognition is presented in Sec. VI. In Sec. VII, we offer a summary and conclusion.

II. Pseudoinverse Associative Processor Formulation

Denoting the keys and recollections as the vectors $\mathbf{x}_k$ (N-dimensional) and $\mathbf{y}_k$ (K-dimensional), respectively, where $k = 1,\ldots,M$, the vectors $\mathbf{x}_k$ and $\mathbf{y}_k$ form an associated key/recollection pair (there are M such pairs). We desire a $K \times N$ matrix $\mathbf{M}$ satisfying

$$\mathbf{y}_k = \mathrm{sgn}(\mathbf{M}\mathbf{x}_k), \tag{1}$$

for $k = 1,\ldots,M$, where $\mathrm{sgn}(\mathbf{M}\mathbf{x}_k)$ indicates that a signum function is applied to each vector element ($\mathrm{sgn}(x) = 1$ if $x \geq 0$ and $\mathrm{sgn}(x) = -1$ otherwise). Defining matrices $\mathbf{X}(N \times M)$ and $\mathbf{Y}(K \times M)$ with the key and recollection vectors as their columns, Eq. (1) can be rewritten as

$$\mathbf{Y} = \mathrm{sgn}(\mathbf{M}\mathbf{X}). \tag{2}$$

It is useful to distinguish between autoassociative processors (AAPs), in which $\mathbf{Y} = \mathbf{X}$, and heteroassociative processors (HAPs). Autoassociative processors are used for restoring partial or noisy inputs. Our major concern is HAPs since they are useful for decisions and pattern recognition. It is well known that the pseudoinverse AAP degenerates to the identity matrix when the *rows* of $\mathbf{X}$ are linearly independent,[5] which is likely to occur when $M > N$. Although this is clearly not a useful processor, it does correctly recall exact key inputs, and the H-K algorithm cannot improve on it. This is another reason why this paper considers only HAPs.

A solution of Eq. (2) is[5]

$$\mathbf{M} = \mathbf{Y}\mathbf{X}^{+}, \tag{3}$$

where $\mathbf{X}^{+}$ is the pseudoinverse of $\mathbf{X}$. If the key vectors are linearly independent, then Eq. (3) is guaranteed to satisfy Eq. (2) exactly. We find linearly independent key vector requirements to be unrealistic. When the key vectors are linearly dependent, the solution in Eq. (3) is approximate. This is guaranteed to be the case when $M > N$. Such an approximate solution is useful and allows a larger storage capacity ($M > N$). Only limited attention has been given to such cases.[1,6,7] We will refer to Eq. (3) as an exact pseudoinverse AP (when $M \leq N$ and the keys are linearly independent) and as an approximate pseudoinverse AP (when $M > N$).

To compute the pseudoinverse, we use the singular value decomposition (SVD) approach[21] because it can be used for either linearly independent or dependent keys and because it allows us to improve the AP's noise performance by a method that will now be explained. The conventional pseudoinverse HAP recalls noisy key vectors poorly when $M \approx N$ (with much better performance occurring when $M < N$ and $M > N$).[6] A recent paper[7] explains this phenomenon and shows that to optimize recall accuracy for a particular noise variance σ^2, all singular values μ_i satisfying

$$\mu_i < \sqrt{M}\sigma \tag{4}$$

should be set to zero. Then the memory matrix is computed by[7]

$$\mathbf{M} = \mathbf{Y}\tilde{\mathbf{X}}^{+}, \tag{5}$$

where $\tilde{\mathbf{X}}$ is the key matrix $\mathbf{X}$ with small singular values set to zero. For realistic σ values, this method causes only a small decrease in the recall accuracy for exact key vector inputs when $M \approx N$, and significantly improves recall accuracy when noise is present. The method is attractive since it does not alter the pseudoinverse HAP's performance for $M \ll N$ and $M \gg N$. We use this approach in our simulations described in Sec. V, which confirm the above statements. We note that it is well known[21] that very small singular values (e.g., 10^{-4}) should always be set to zero to avoid problems of numerical instability. The method explained above differs from this, in that the threshold for zeroing the singular values is given as a function of M and σ (as opposed to selecting it arbitrarily) and that the singular values set to zero can exceed 10^{-4} by orders of magnitude (e.g., if $M = 50$ and $\sigma = 0.1$, the threshold for μ is 0.71).

III. Ho-Kashyap Associative Processors

The H-K AP has a larger storage capacity than the pseudoinverse AP because it requires that the key vectors be only linearly separable for perfect recall, rather than linearly independent, as the pseudoinverse AP requires. Since linear separability is a looser restriction than linear independence, the H-K AP in many cases can perfectly store linearly dependent keys. Before presenting the H-K algorithm, we first describe how linear separability applies to APs.

A. Linear Separability and the H-K Algorithm

Recall that the columns of $\mathbf{Y}$ are the recollection vectors for the different key vectors. Hence, row i of $\mathbf{Y}$ gives the desired values of the ith output element for the different key vectors. Each row of the AP matrix, with its threshold value, forms a linear discriminant function (LDF) that separates the N-dimensional input space with a hyperplane into two classes, those key vectors for which element i of the recollection vector is -1 and those for which it is $+1$. The locations of the ± 1 elements in row i of $\mathbf{Y}$ denote these two classes for that row. If these two classes can be separated with a hyperplane, then they are linearly separable and there exists an LDF that will give perfect recall for that output recollection element. The pseudoinverse AP minimizes the squared error, but is not guaranteed to give perfect recall even if the K class groupings (one for each output recollection vector element) are all linearly separable.[22] The Ho-Kashyap algorithm iteratively computes an LDF (i.e., one row of $\mathbf{M}$) that will correctly classify two classes if they are linearly separable.[22] If they are not linearly separable, the algorithm will still converge to a minimum squared error solution, and will indicate that the classes are not linearly separable and which output vector elements may not be correct.

B. Ho-Kashyap APs

The H-K algorithm in a new matrix version for AP synthesis is noted in Table I. We begin with an estimate of $\mathbf{M}$ from the pseudoinverse (step 1). The pseudoinverse memory is only an estimate because it is only an approximate solution for $M > N$. We modify $\mathbf{Y}$ (step 4) and $\mathbf{M}$ (step 1) in successive iterations. If the pseudoinverse is exact (i.e., the keys are linearly independent) then no modifications will be made. The H-K algorithm improves the pseudoinverse memory when the keys are linearly dependent. In step 2, we calculate the error matrix $\mathbf{E}$, which gives the errors between the actual and desired outputs. The matrix $\mathbf{S}$ in step 3 contains the signs of the $\mathbf{Y}$ elements, $\otimes$ denotes Hadamard (pointwise) multiplication, and the subscript n is the iteration index. In step 3, we use $\mathbf{S}$ to form a modified error matrix $\mathbf{E}'$. This matrix equals $\mathbf{E}$ except that all $\mathbf{E}$ elements that differ in sign from the corresponding elements in $\mathbf{Y}$ are set to 0. This ensures that none of the $\mathbf{Y}$ elements change sign (we assume initial bipolar binary $\mathbf{Y}$ values) when $\mathbf{E}'$ is added to $\mathbf{Y}$ in step 4 to produce an updated $\mathbf{Y}$. The signs of $\mathbf{Y}$ cannot be allowed to change sign because the signs determine on and off recollection vector elements. Step 5 then returns the algorithm to step 1 where $\mathbf{M}$ is updated. Once $\mathbf{E}'_n = \mathbf{0}$, the algorithm has converged (convergence is guaranteed whether the keys are linearly separable or not). If a row of $\mathbf{E}_n$ equals $\mathbf{0}$ then that row's dichotomy (grouping into two classes) is linearly separable; otherwise it is not. In actual application, the algorithm can also be stopped if $\mathbf{M}$ correctly recalls all of the key vectors.

C. Most Reliable Recollection Vector Elements

Since the final $\mathbf{E}$ indicates which output elements give perfect recall for the key vector inputs, the H-K algorithm automatically provides information about which output elements are the most reliable. If the

Table I. Ho-Kashyap AP Algorithm

Step	Operation
1	$\mathbf{M}_n = \mathbf{Y}_n\mathbf{X}^{+}$
2	$\mathbf{E}_n = \mathbf{M}_n\mathbf{X} - \mathbf{Y}_n$
3	$\mathbf{E}'_n = \mathbf{S} \otimes \frac{1}{2}[(\mathbf{S} \otimes \mathbf{E}_n) + \lvert\mathbf{S} \otimes \mathbf{E}_n\rvert]$
4	$\mathbf{Y}_{n+1} = \mathbf{Y}_n + 2\rho\mathbf{E}'_n, \quad 0 < \rho < 1$
5	If $\mathbf{E}'_n \neq \mathbf{0}$ go to 1.

data are linearly separable, $\mathbf{E}$ and $\mathbf{E}'$ will be all zero (and the new $\mathbf{Y}$ and $\mathbf{M}$ achieve this linear separation). If $\mathbf{E}'$ is all zero and $\mathbf{E}$ is not, at least one row of $\mathbf{Y}$ is not linearly separable (but the resultant $\mathbf{M}$ is better than the approximate pseudoinverse solution, in that it has a lower squared error). The rows of $\mathbf{E}$ and $\mathbf{E}'$ with all zero elements denote the output elements that are reliable. This allows us to consider the reliable outputs first and then the other elements with a reduced confidence algorithm.

D. Robust Ho-Kashyap AP

Our basic H-K AP (Table I) uses the exact or approximate pseudoinverse AP as an initial solution and then refines it. Since the pseudoinverse is used in the basic Ho-Kashyap algorithm, the resulting memory will suffer the same recall deficiencies as the pseudoinverse memory when $M \approx N$. We therefore propose that $\tilde{\mathbf{X}}^+$ in Eq. (5) be used instead of $\mathbf{X}^+$ in Table I. We call this the robust H-K AP. This combination of the H-K algorithm and setting small eigenvalues to zero is quite new.

Since $\tilde{\mathbf{X}}^+ \neq \mathbf{X}^+$ in all cases, the robust H-K AP is not guaranteed to find a linearly separable solution if one exists. However, this is not a major problem. We have $\tilde{\mathbf{X}}^+ \neq \mathbf{X}^+$ only when $M \approx N$, when the keys are likely to be easy to linearly separate. Thus, we are still likely to be able to find the solution, even though $\tilde{\mathbf{X}}^+$ differs slightly from $\mathbf{X}^+$. As M grows larger, the keys tend to become harder to linearly separate, but $\tilde{\mathbf{X}}^+$ tends to become identical to $\mathbf{X}^+$, which guarantees that a linearly separable solution will be found if one exists. These comments are confirmed by the simulations of Sec. V.B. Since $\tilde{\mathbf{X}}^+ \neq \mathbf{X}^+$ in some cases, we must also find the conditions under which the robust algorithm is guaranteed to converge. We have shown (Appendix A) that its convergence conditions are identical to those for the original algorithm, that is, $0 < \rho < 1$ in step 4 in Table I.

E. Error Correcting Ho-Kashyap AP Algorithm

We now mention the use of an error correcting H-K algorithm[23] that can be used to produce a new error correcting H-K AP algorithm, which does not require an initial $\mathbf{X}^+$. Because of its error correcting nature and the fact that SVD is not used, we expect it to tolerate lower accuracy than the first H-K algorithm (Table I). Hence it appears attractive for optical implementation. The algorithm updates $\mathbf{Y}$ and $\mathbf{M}$ using

$$\begin{aligned} \mathbf{Y}_{n+1} &= \mathbf{Y}_n + \mathbf{E}_n' \\ \mathbf{M}_{n+1} &= \mathbf{M}_n + \rho(\mathbf{S} \otimes |\mathbf{E}_n'|)\mathbf{X}^T\mathbf{R}, \end{aligned} \tag{6}$$

where $\mathbf{R}$ can be any positive definite $N \times N$ matrix. The simplest choice is $\mathbf{R} = \mathbf{I}$.

IV. Optical Implementation

The recall operation of the pseudoinverse and H-K APs can be performed by the standard optical analog matrix-vector multiplier shown in Fig. 1. The optical system is attractive for its high speed parallel computing power. The system operates as follows. The P_1 input plane contains N point modulators with light outputs proportional to $\mathbf{x}$. Each element of $\mathbf{x}$ uniformly illuminates one column of the memory $\mathbf{M}$, a transmittance array of $K \times N$ elements at P_2, and the light leaving P_2 is integrated horizontally onto K detectors at P_3. The detector output is the matrix-vector product $\mathbf{y} = \mathbf{M}\mathbf{x}$. Passing this through a signum function gives the desired final output $\mathbf{y} = \text{sgn}(\mathbf{M}\mathbf{x})$. The matrix $\mathbf{M}$ will be bipolar. We note that a variety of techniques have been developed for optically representing bipolar data.[24-27]

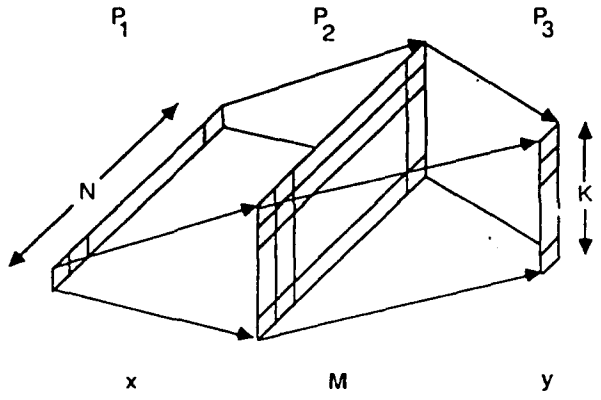


Fig. 1. Analog optical matrix-vector multiplier for associative processor recall.

V. General Memory Ho-Kashyap Associative Processors

We first review theoretical work and then report our simulation results.

A. Theoretical Results

Classic theoretical results allow us to estimate the storage capacity of the H-K AP. The results assume that the M N-dimensional key vectors are in general position. For a group of vectors to be in general position, no subset of N vectors can be linearly dependent. Thus, restricting vectors to be in general condition is a looser condition than linear independence. There are 2^M possible dichotomies of these vectors. The fraction of these that are linearly separable is[10,22]

$$f(M,N) = \begin{cases} 1 & M \leq N \\ 2^{1-M} \sum_{i=0}^{N-1} \binom{M-1}{i} & M > N. \end{cases} \tag{7}$$

When the keys cannot be assumed to be in general position, Eq. (7) is an upper bound. We extend Eq. (7) to an associative memory formulation (with K-element recollection vectors) by finding the fraction of groups of K dichotomies that are all linearly separable. This gives the fraction of all possible $\mathbf{Y}$ matrices that can be correctly recalled in an H-K AP. Our fraction is Eq. (7) raised to the Kth power:

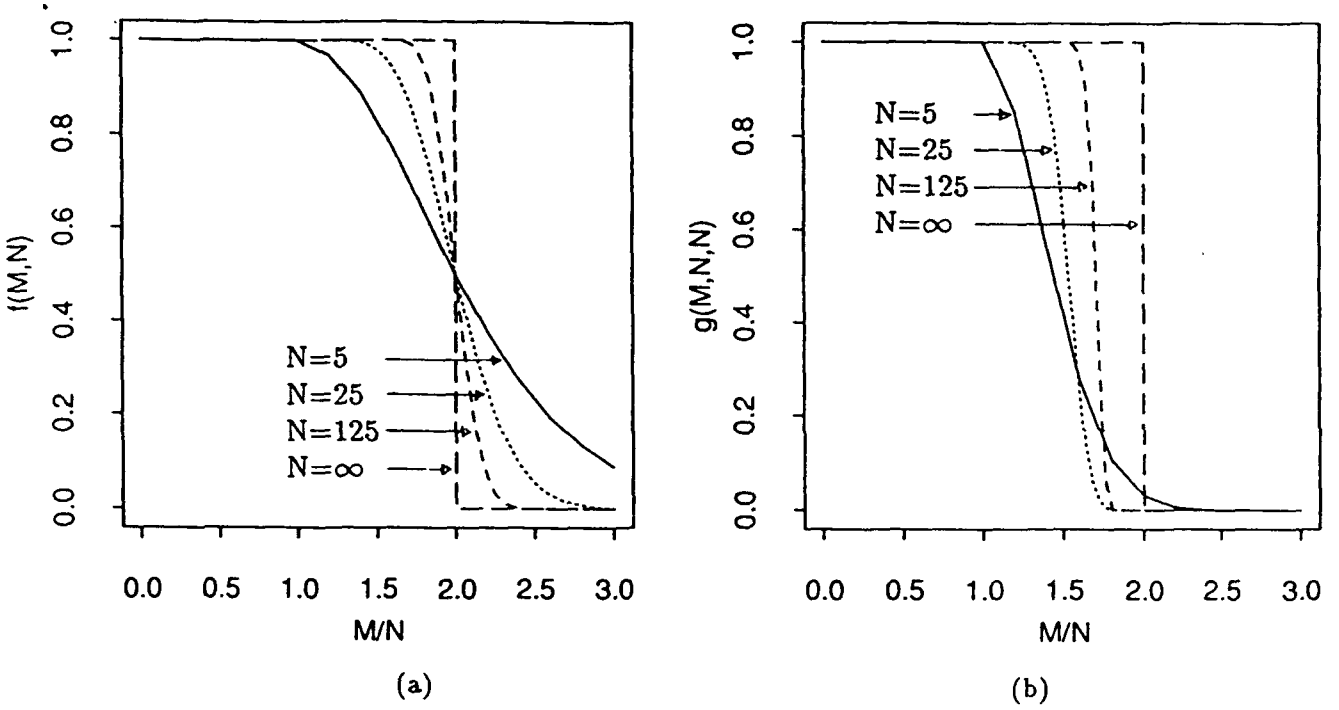


Fig. 2. Fraction of (a) all dichotomies of M N-dimensional vectors that are linearly separable, and (b) all groups of K dichotomies of M N-dimensional vectors that are linearly separable.

$$g(M,N,K) = \begin{cases} 1 & M \le N \\ 2^{K-MK}\left[\sum_{i=0}^{N-1}\binom{M-1}{i}\right]^K & M > N. \end{cases} \quad (8)$$

The asymptotic limit (for $K = N$) as $N \to \infty$ is (modified from Ref. 28)

$$g(M,N,N) = \begin{cases} 1 & M/N < 2 \\ 0 & M/N \ge 2. \end{cases} \quad (9)$$

Thus, the maximum storage for an H-K AP in a general memory application (random keys and recollections) is

$$M = 2N. \quad (10)$$

This asymptotic limit is not achievable with finite length (N) key vectors. If N is increased, M can be increased accordingly (at the cost of increased memory size). The value of N can be increased with higher order APs[10,29,30] or by forming random combinations of the original key vector elements.[11,31] We do not consider these approaches, but our work in increasing the AP capacity as a function of N applies to the transformed key vectors produced by these methods.

Figure 2(a) plots Eq. (7) vs M/N. It shows the probability (the fractional amount) that one row of Y specifies a linearly separable grouping of key vectors. Figure 2(b) plots Eq. (8) vs M/N for $K = N$. It shows the probability that all K rows of Y specify linearly separable groupings. As seen, the maximum storage capacity of $M = 2N$ cannot be achieved except with infinite N, but Eq. (8) allows us to estimate the storage capacity for finite N. For example, for $N = 125$, we see from Fig. 2(b) that the probability that all rows of Y designate linearly separable groupings is essentially 1 for a large storage capacity $M \le 1.5N$. Even if a row of Y does not specify a linearly separable grouping, it is still possible for the corresponding output element to be correct much of the time.

B. Ho-Kashyap and Pseudoinverse General Memory Simulations

We now test random H-K APs for agreement with the above theory and for comparison to pseudoinverse APs. We use $N = 50$ element key vectors, $K = N = 50$ element recollection vectors and vary M/N. (This general memory uses equal key and recollection vector dimensions. If the AP outputs were labels used to read out high dimensional recollection vectors from a larger second stage standard addressable memory, the AP outputs would be of dimension $K < N$ and the memory matrix would be smaller.) When $M/N > 1$, the key vectors are automatically linearly dependent. All key vectors were randomly chosen and uniformly distributed over -1 to $+1$. Each bipolar binary recollection vector element was chosen randomly to be -1 or $+1$. For each M/N value, ten X and ten Y matrices were generated. Our results are averaged over the ten resulting memory matrices for each M/N value tested. The H-K synthesis algorithm used $\rho = 0.5$ and was

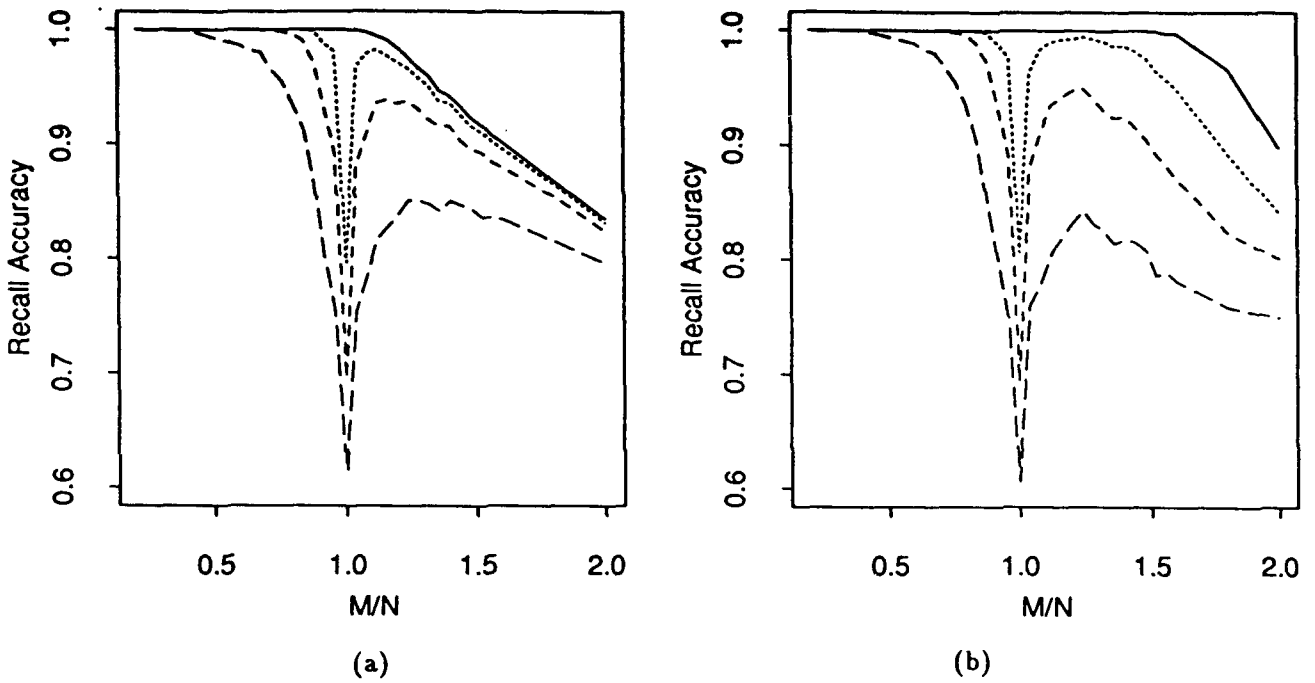


Fig. 3. Recall accuracy vs M/N for exact and noisy key vector inputs using (a) pseudoinverse associative memory and (b) basic Ho-Kashyap associative memory.

limited to a maximum of 1000 iterations. The algorithm was also stopped when the memory perfectly recalled the exact key inputs or when $\mathbf{E}' = \mathbf{0}$. To test each AP, we used the M key vectors with four levels of additive zero-mean Gaussian noise: $\sigma = 0.0, 0.05, 0.1, 0.2$. The last three nonzero noise levels correspond to signal-to-noise ratios of 21, 15, and 9 dB, respectively. Since the key vector elements were bounded by -1 and $+1$, we bounded the noisy inputs to be within the same limits. This limiting only slightly improved the recall results (less than 1% improvement). The recall accuracy (percentage of correct output elements) was computed for each noise level. Figure 3 shows the results for the standard pseudoinverse AP [Fig. 3(a)] and the results for our basic H-K AP [Fig. 3(b)], for M/N ratios of 0.2, 0.4, 0.6 to 1.6 in 0.04 increments, 1.8 and 2.0. These results show improved performance and storage capacity for the H-K vs the pseudoinverse AP. For discussion purposes, we consider an AP to be useful if its recall accuracy for exact key inputs exceeds 0.999. The recall accuracy of the pseudoinverse AP exceeds 0.999 for exact key inputs only up to $M = 1.04N$, and degrades for $M > 1.04N$. The H-K AP exceeds 0.999 recall for $M \leq 1.52N$ for inputs with no noise. This is a 45% improvement in the capacity of the H-K over the pseudoinverse AP. Thus, the H-K AP performs significantly better than the pseudoinverse AP. Its improvement over the correlation HAP ($M \approx 0.15N$)[1] is a factor of 10 or 900% better performance. We note that in all cases, the performance in noise degrades when $M \approx N$ (as expected for APs based on the pseudoinverse). For noisy inputs when $M > N$, the H-K AP performs better than the pseudoinverse AP at low noise levels ($\sigma = 0.05$ for all M/N and $\sigma = 0.1$ for $M \leq 1.48N$). Although the pseudoinverse AP performs better at higher noise levels ($\sigma = 0.1$ for $M \geq 1.52N$ and $\sigma = 0.2$ for $M > N$), the recall accuracy is low (<90%) and, thus, this difference is not of concern (since neither AP performs very well for these noise levels). For the specific case of $M = 1.52N$, the H-K AP recall was 0.05 higher than the pseudoinverse AP recall for $\sigma = 0.05$; the two recall accuracies were nearly identical for $\sigma = 0.1$; and both recall accuracies were low (<90%) for $\sigma = 0.2$ when $M > N$. Thus, neither AP may be suitable for inputs with a large amount of noise ($\sigma = 0.2$) at high storage capacities ($M > N$). But, for reasonable noise and performance, the H-K AP is preferable and can be used when $M > N$.

We now consider the use of our robust H-K AP to improve noise performance when $M \approx N$. For comparison, we apply the robust algorithm with small eigenvalues removed to both the pseudoinverse AP and the H-K AP. We set $\sigma = 0.1$ for the singular value threshold expressed by Eq. (4). The exact value of this threshold is not critical, since our choice for the threshold also gives good performance for $\sigma = 0.05$ and $\sigma = 0.20$. The results are shown in Figs. 4(a) and 4(b) respectively, with M/N varied from 0.6 to 1.6 in 0.04 increments. As seen, both APs avoid the severe drop in performance (when $M \approx N$) in Fig. 3 for noisy inputs. For the robust APs with exact key inputs, we obtained:

$P_c \geq 99.9\%$ for $M \leq 0.84N$ for the robust pseudoinverse AP,

$P_c \geq 99.9\%$ for $M \leq 1.52N$ for the robust H-K AP,

80% storage improvement with robust H-K AP over robust pseudoinverse AP.

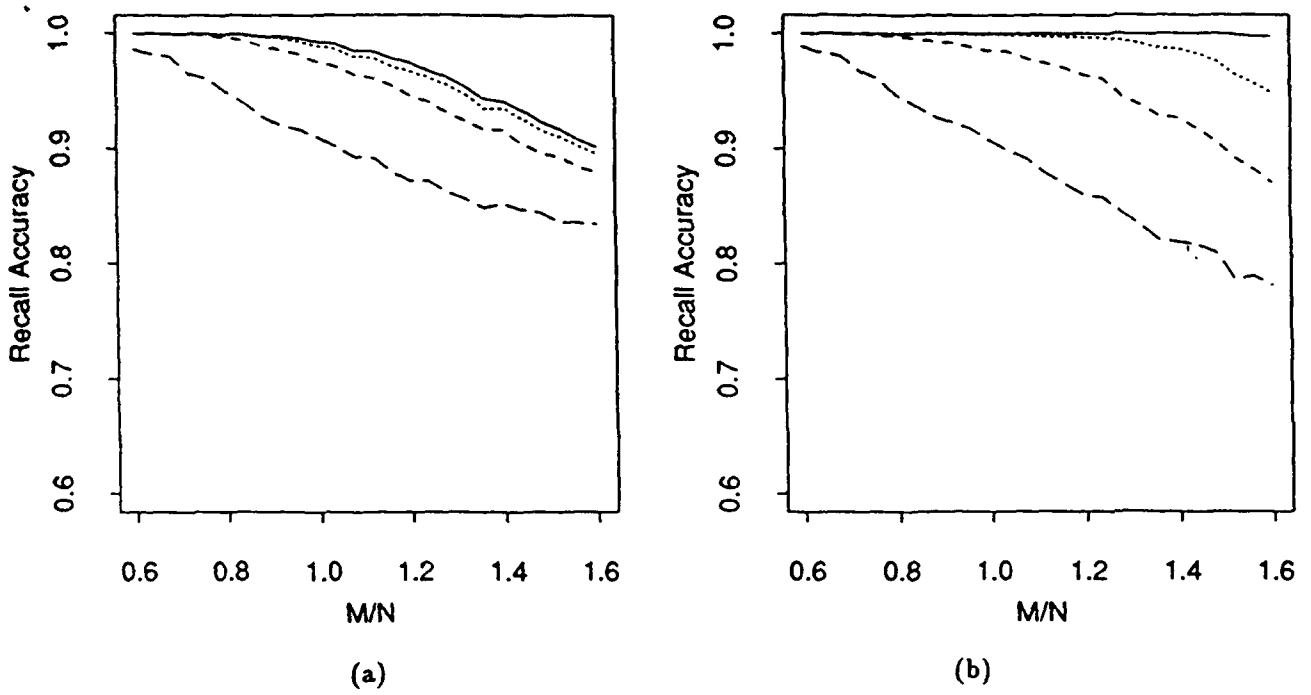


Fig. 4. Recall accuracy vs M/N for exact and noisy key vector inputs using (a) robust pseudoinverse associative memory and (b) robust Ho-Kashyap associative memory.

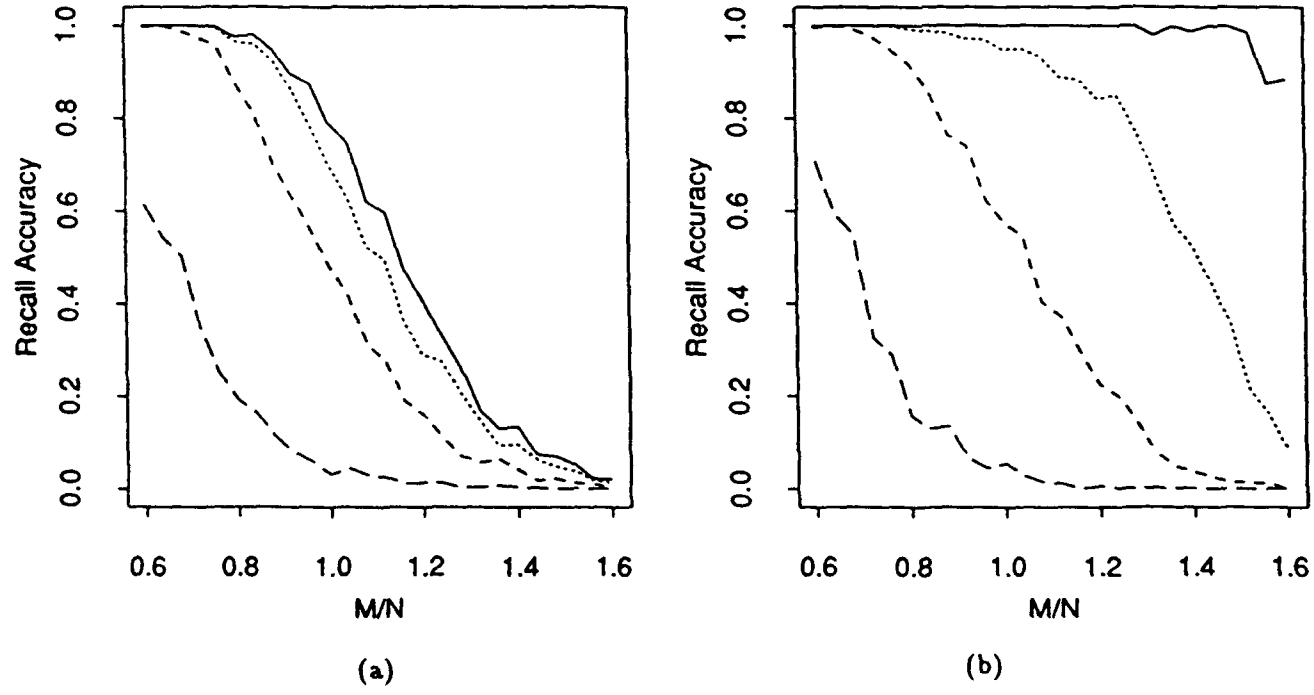


Fig. 5. Fraction of output vectors that are completely correct vs M/N for exact and noisy key vector inputs using (a) robust pseudoinverse associative memory and (b) robust Ho-Kashyap associative memory.

The robust pseudoinverse AP performs worse than the standard pseudoinverse AP with no noise, but gives much better recall when noise is present and $M \approx N$. The robust H-K AP performs best for exact key inputs and inputs with low noise. The differences in the robust H-K and pseudoinverse AP performance for noisy inputs when $M > N$ are the same as described above for the standard H-K and pseudoinverse APs. This is expected, since omitting small singular values only affects results when $M \approx N$. Hence, the recall accuracy curves away from $M = N$ are the same for the robust and standard algorithms. Figure 5 plots the fraction of M recollection vectors that are completely correct for the robust pseudoinverse [Fig. 5(a)] and the robust H-K [Fig. 5(b)] APs vs M/N for different amounts of noise. Again, the robust H-K AP is preferable when the recall accuracy is good (above 90%).

In Table II, we show the rank of $\tilde{\mathbf{X}}$ (the key matrix with small eigenvalues set to zero) for $N = 50$ and for various values of M. The entry $\min\{M,N\}$ is the minimum of M and N and indicates what the rank of $\tilde{\mathbf{X}}$ would be if $\tilde{\mathbf{X}}$ were of full rank. The original $\mathbf{X}$ is full rank for all M/N. By comparing the $\min\{M,N\}$ and rank entries, we see when small eigenvalues are omitted. For $M/N = 0.8$, we omit an average of 0.7 eigenvalues and for $M/N = 1.0$, we omit an average of 5.7. For $M/N \geq 1.52$, no eigenvalues are set to zero and $\tilde{\mathbf{X}} = \mathbf{X}$ is of full rank. Thus, the robust H-K AP differs from the standard H-K AP for $0.76N \leq M \leq 1.48N$. In all cases in this region where the standard H-K AP perfectly recalled all exact key inputs, the robust H-K AP also gave perfect recall accuracy. This experimental evidence confirms the argument of Sec. III.D that the robust H-K algorithm is highly likely to find a linearly separable solution if one exists.

Table III shows the number of robust H-K iterations used for different M/N ratios (with $N = 50$). For $M/N < 0.8$, we see that the pseudoinverse is exact (no singular values are set to zero) since the H-K algorithm does not iterate. For $M/N \geq 0.8$, the robust pseudoinverse sets some singular values to zero and the H-K algorithm is used to restore the recall accuracy for noiseless key vectors. The number of H-K iterations required increases with M/N because the keys become more difficult to linearly separate.

We note good agreement between theory and tests. For both the standard and robust H-K APs, all APs tested for $M/N \leq 1.28$ and $M/N = 1.44$ were linearly separable. Of the ten APs tested for each other M/N value, at least one of each set was not linearly separable. With $N = K = 50$, Eq. (8) predicts that the transition from ten linearly separable memories to at least one linearly nonseparable memory will occur between roughly $M/N = 1.34$ and $M/N = 1.56$, with the probability that all ten memories are linearly separable being 0.99 at $M/N = 1.34$ and 0.05 at $M/N = 1.56$. The experimental transition occurs at the lower end of the theoretical transition. This is to be expected since the theoretical transition is an upper bound due to its general position assumption. The two transitions still agree reasonably well. Thus, Eq. (8) allows us to estimate capacity of the H-K AP for finite N.

VI. Pattern Recognition Ho-Kashyap Associative Processors

This section presents a comparison of the pseudoinverse and Ho-Kashyap APs in a two-class distorted aircraft pattern recognition problem. We consider two classes (Phantom and DC-10) of 128 × 128 pixel aircraft imagery. Nominal views of these aircraft are shown in Figs. 6(a) and 6(b). As our key vector representation space, we use thirty-two wedge samples of the Fourier transform (in half of the transform plane). The wedge feature space provides scale invariance (when the wedge samples are normalized) and shift invariance, and is easily generated optically.[32] In-plane image rotations cause the wedge samples to circularly shift. We consider the case when the aircraft is moving and that tracking information provides the location of the aircraft's nose. This information

Table II. Average Rank of the Modified Key Matrix $\tilde{\mathbf{X}}$ for Different M/N, with $\sigma = 0.1$ for the Singular Value Threshold

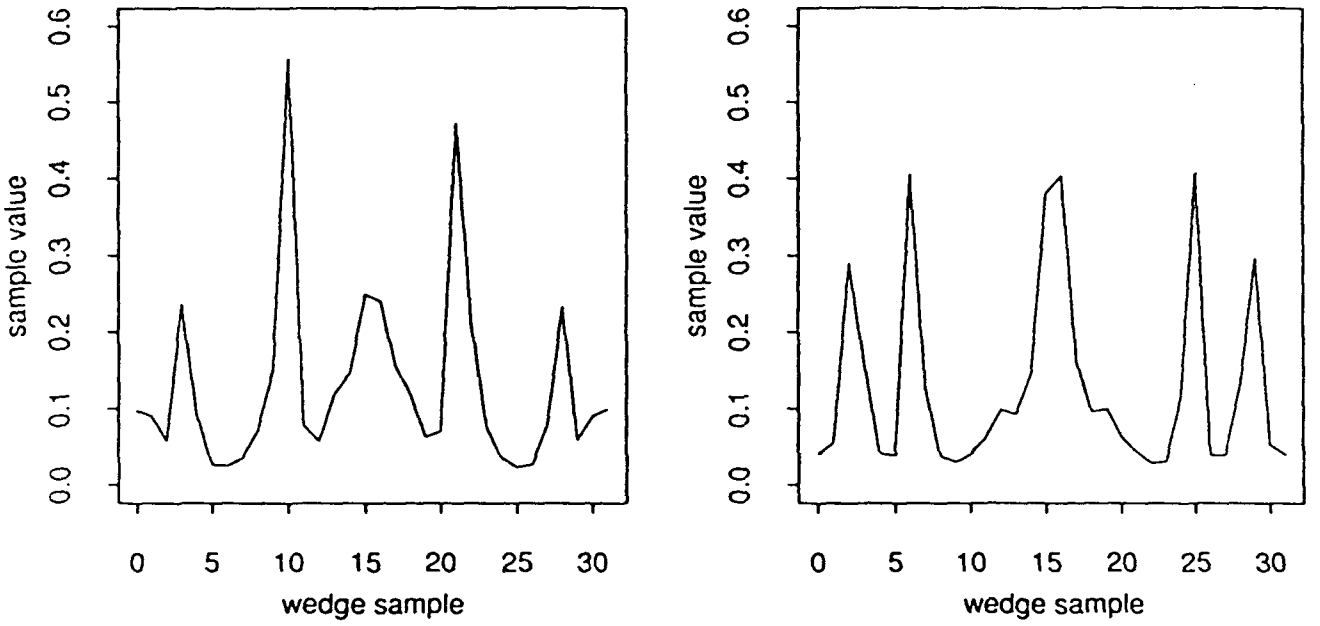
M/N	0.60	0.64	0.68	0.72	0.76	0.80	0.84	0.88	0.92	0.96	1.00	1.04	1.08
$min\{M,N\}$	30	32	34	36	38	40	42	44	46	48	50	50	50
rank	30.0	32.0	34.0	36.0	37.7	39.3	40.6	41.5	42.5	43.4	44.3	45.4	45.5
M/N	1.12	1.16	1.20	1.24	1.28	1.32	1.36	1.40	1.44	1.48	1.52	1.56	1.60
$min\{M,N\}$	50	50	50	50	50	50	50	50	50	50	50	50	50
rank	46.6	47.1	48.0	48.3	48.9	49.1	49.1	49.8	49.9	49.9	50.0	50.0	50.0

Table III. Average Number of Iterations Required by Robust Ho-Kashyap Algorithm for Different M/N

M/N	0.60	0.64	0.68	0.72	0.76	0.80	0.84	0.88	0.92	0.96	1.00	1.04	1.08
iterations	0	0	0	0	0	1	1	2	3	4	5	9	17
M/N	1.12	1.16	1.20	1.24	1.28	1.32	1.36	1.40	1.44	1.48	1.52	1.56	1.60
iterations	14	29	13	25	41	81	188	232	108	310	643	714	880

(a) (b)

Fig. 6. Images used in the aircraft recognition problem: (a) Phantom and (b) DC-10 aircraft.



(a) (b)

Fig. 7. Wedge samples for the (a) Phantom and (b) DC-10 images in Fig. 6.

allows the wedge samples from an unidentified aircraft to be circularly shifted so that they align properly with the training vectors. Thus, for moving aircraft this feature space is rotation (in-plane), scale and shift invariant. Thus, we do not test these invariances. Rather, we consider aircraft with out-of-plane distortions. Figures 7(a) and 7(b) show the feature vectors corresponding to the images in Fig. 6. The positions of the peaks correspond to the angles of the edges of the object and the peak values correspond to the lengths of the edges. This interpretation gives the wedge feature space an intuitive appeal. We augment each of the key vectors with a 1. This increases the dimension of the key vector hyperspace from N to $N + 1$ and does not require the separating hyperplanes to pass through the origin, as would occur otherwise. This technique is well known[10,22] and in APs is equivalent to varying the thresholds on the output elements.[14] The recollection vectors are of dimension $K = 1$ with values of $+1$ for the Phantom and -1 for the DC-10.

Two training sets were used. The first consists of 882 key vectors for the two aircraft rotated in pitch and roll between $\pm 50°$ at 5° increments. The second consists of 2178 key vectors for the two aircraft rotated in pitch and roll between $\pm 80°$ at 5° increments. Thus, we consider APs with

N = 33-dimensional key vectors,
M = 882 and 2178 key/recollection pairs,
K = 1-dimensional recollection vectors.

Both cases represent linearly dependent key vectors, with large capacities $M = 25N$ and $M = 65N$ respectively. An H-K AP was produced using the same stopping criteria as in Sec. V. (We did not test the robust H-K algorithm since the test vector distortions are not easily quantified into an equivalent noise σ.) The first case ($M = 882$) was found to be a linearly separable problem, since the H-K algorithm gave 100% correct classification after thirty-two iterations. As shown in Table IV, the H-K AP yields perfect performance on the training set, whereas the pseudoinverse AP does not. For test data in Table IV, we used 800 aircraft (not present in the training set) with pitch and roll varied between ±47.5° at 5° increments (i.e., at least 2.5° different in pitch and roll from the training data). The H-K AP also gives perfect performance for these inputs and better performance than the pseudoinverse AP. The second case ($M = 2178$) represents a linearly nonseparable problem, as shown in Table V. The H-K algorithm was stopped at eighty iterations since $\mathbf{E}' = \mathbf{0}$ then. However, $\mathbf{E} \neq \mathbf{0}$, and thus the algorithm indicated that the keys were not linearly separable. The test data in Table V used 2048 aircraft with pitch and roll varied between ±77.5° at 5° increments. We see that the H-K AP gives excellent performance in both cases.

VII. Summary and Conclusion

We have shown that the Ho-Kashyap associative processor has a larger storage capacity than the pseudoinverse processor and that it can store linearly dependent key vectors more accurately than the pseudoinverse processor. We have detailed a new robust Ho-Kashyap processor to improve the noise performance of the H-K AP. This new processor allows operation on linearly dependent key vectors, achieves much better storage ($M \approx 2N$ for general memory applications), and significantly improves noise performance when $M \approx N$. (The last advantage is due to incorporating Murakami and Aibara's technique.[7]) For $N = 50$ element key vectors, we showed: 100% recall accuracy for our Ho-Kashyap general memories for $M < 1.5N$ and 99.7% accuracy with $M = 1.6N$; nearly 900% larger storage capacity than a correlation AP; 40% larger storage capacity than the pseudoinverse memory; and 90% improved noise performance when $M \approx N$. Our pattern recognition case study showed 3-D distortion invariance and excellent (>93%) recall accuracy for large $M = 25N$ and $M = 65N$ cases with linearly dependent key vectors. We have discussed an optical architecture for implementing H-K recall. The error-correcting AP algorithm that we propose appears attractive for optical AP synthesis because of its expected low dynamic range requirements.

Table IV. Misclassification Results for Pseudoinverse and Ho-Kashyap Memories with ±50° Training Set

% Misclassified		
Training Set:	Pseudoinverse	0.68
	H-K AP	0.00
Test Set:	Pseudoinverse	0.25
	H-K AP	0.00

Table V. Misclassification Results for Pseudoinverse and Ho-Kashyap Memories with ±80° Training Set

% Misclassified		
Training Set:	H-K AP	7.0
Test Set:	H-K AP	6.1

Appendix A: Convergence Proof for Robust Ho-Kashyap Algorithm

We prove that the robust H-K algorithm converges when $0 < \rho < 1$. Without loss of generality, we consider the case where $K = 1$ (i.e., $\mathbf{Y}$ and $\mathbf{M}$ are row vectors). To simplify notation, let $\mathbf{m}$ be an $N \times 1$ column vector that equals $\mathbf{M}^T$, and $\mathbf{b}$ be an $M \times 1$ column vector that equals $\mathbf{Y}^T$, and let $\mathbf{Z} = \mathbf{X}^T$, and $\tilde{\mathbf{Z}} = \tilde{\mathbf{X}}^T$. We also multiply all key vectors belonging to the second class by -1. This makes the desired outputs $\mathbf{b}$ all positive. (Initially, $\mathbf{b}$ is all $+1$ and during the iterations the output elements change but remain positive.) The robust H-K algorithm is now given by

Step	Operation	
1	$\mathbf{m}_n = \tilde{\mathbf{Z}}^+ \mathbf{b}_n$,	(A1)
2	$\mathbf{e}_n = \mathbf{Z}\mathbf{m}_n - \mathbf{b}_n$,	(A2)
3	$\mathbf{e}'_n = (1/2)(\mathbf{e}_n + \lvert\mathbf{e}_n\rvert)$,	(A3)
4	$\mathbf{b}_{n+1} = \mathbf{b}_n + 2\rho\mathbf{e}'_n$,	(A4)
5	If $\mathbf{e}'_n \neq 0$ go to 1.	(A5)

Step 3 sets all negative $\mathbf{e}$ elements to 0. The modified key matrix $\tilde{\mathbf{Z}}$ is used in the pseudoinverse in Eq. (A1) to improve noise performance when $M \approx N$. The unmodified key matrix $\mathbf{Z}$ is used to compute the error in Eq. (A2) because we want all the noiseless keys to be correctly recalled.

The proof follows the same steps as in Ref. 22 for the original H-K algorithm. However, the proof in Ref. 22 requires that $M \geq N$ and that $\mathbf{X}$ be full rank. These are valid assumptions for overdetermined PR applications, but not for the general memory application. Our proof for the robust algorithm makes no such assumptions.

The proof makes use of the facts that $\mathbf{Z}\tilde{\mathbf{Z}}^+$ is symmetric, positive semidefinite and idempotent (i.e., the square of the matrix equals itself). We show these properties using the SVD[21] of $\mathbf{Z}$, which is given by $\mathbf{Z} = \mathbf{U}\Sigma\mathbf{V}^T$, where $\mathbf{U}$ and $\mathbf{V}$ are $M \times R$ and $N \times R$ matrices (R is the rank of $\mathbf{Z}$) with orthonormal columns, and Σ is an $R \times R$ diagonal containing the R singular values of $\mathbf{Z}$. Note that $\mathbf{U}^T\mathbf{U} = \mathbf{V}^T\mathbf{V} = \mathbf{I}$. The pseudoinverse is given by $\mathbf{Z}^+ = \mathbf{V}\Sigma^+\mathbf{U}^T$, where Σ^+ is a diagonal matrix containing the reciprocals of the singular values (except for singular values equal to zero, which remain zero). The modified $\mathbf{Z}$ is given by $\tilde{\mathbf{Z}} = \mathbf{U}\tilde{\Sigma}\mathbf{V}^T$, where $\tilde{\Sigma}$

identical to Σ except that the small singular values ave been set to 0. We denote the rank of $\tilde{\Sigma}$ as $\tilde{R}$. We e that $Z\tilde{Z}^+ = U\Sigma V^T V\tilde{\Sigma}^+ U^T = U\tilde{I}U^T$, where $\tilde{I}$ is a iagonal matrix with the first $\tilde{R}$ diagonal elements ual to 1 and the remaining elements equal to 0. learly, $U\tilde{I}U^T$, and hence $Z\tilde{Z}^+$, are symmetric and ositive semidefinite. We also see that $(Z\tilde{Z}^+)(Z\tilde{Z}^+) = U\tilde{I}U^T U\tilde{I}U^T = U\tilde{I}U^T = Z\tilde{Z}^+$, so $Z\tilde{Z}^+$ is idempotent. e now use these matrix properties.

To show that the algorithm converges, we show that $\|e_n\|^2 - \|e_{n+1}\|^2$ is positive. Substituting Eq. (A1) into q. (A2) gives

$$e_n = (Z\tilde{Z}^+ - I)b_n. \tag{A6}$$

ubstituting Eq. (A4) into Eq. (A6) gives

$$e_{n+1} = e_n + 2\rho(Z\tilde{Z}^+ - I)e_n. \tag{A7}$$

aking 1/4 of the squared norm of each side of Eq. (A7) ields

$$\tfrac{1}{4}\|e_{n+1}\|^2 = \tfrac{1}{4}e_n^T e_n + \rho e_n^T(Z\tilde{Z}^+ - I)e_n + \rho^2 e_n'^T(Z\tilde{Z}^+ - I)^T(Z\tilde{Z}^+ - I)e_n, \tag{A8}$$

$$\tfrac{1}{4}(\|e_n\|^2 - \|e_{n+1}\|^2) = \rho e_n^T e_n - \rho e_n^T Z\tilde{Z}^+ e_n - \rho^2 e_n'^T(Z\tilde{Z}^+ - I)^T(Z\tilde{Z}^+ - I)e_n. \tag{A9}$$

he first term on the right hand side of Eq. (A9) equals $\|e_n\|^2$ because the negative e_n elements are multiplied y the e_n elements that are 0. The second term reuces to 0. To show this, substitute Eq. (A6) for e_n nto this term to obtain

$$\begin{aligned} -\rho e_n^T Z\tilde{Z}^+ e_n &= -\rho b_n^T(Z\tilde{Z}^+ - I)Z\tilde{Z}^+ e_n \\ &= \rho b_n^T(Z\tilde{Z}^+ - Z\tilde{Z}^+)e_n = 0, \end{aligned} \tag{A10}$$

here the second equality uses the fact that $Z\tilde{Z}^+$ is dempotent. The third term can be expanded as

$$-\rho^2 e_n'^T[(Z\tilde{Z}^+)^T(Z\tilde{Z}^+) - (Z\tilde{Z}^+)^T - Z\tilde{Z}^+ + I]e_n'^T. \tag{A11}$$

ince $Z\tilde{Z}^+$ is symmetric and idempotent, the term simlifies to

$$-\rho^2 e_n'^T(I - Z\tilde{Z}^+)e_n. \tag{A12}$$

'ith these simplifications, Eq. (A9) can now be rewrit- n as

$$\tfrac{1}{4}(\|e_n\|^2 - \|e_{n+1}\|^2) = \rho(1-\rho)\|e_n\|^2 + \rho^2 e_n'^T Z\tilde{Z}^+ e_n. \tag{A13}$$

he quantity $\|e_n\|^2$ is strictly positive (it can be zero nly when the algorithm has terminated) and the secnd term on the right hand side is nonnegative since $\tilde{Z}^+$ is positive semidefinite. Therefore, the algorithm guaranteed to converge if $\rho(1-\rho) > 0$, or for $0 < \rho <$

References

G. S. Stiles and D.-L. Denq, "A Quantitative Comparison of the Performance of Three Discrete Distributed Associative Memory Models," IEEE Trans. Comput. C-36, 257–263 (1987).

J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," Proc. Natl. Acad. Sci. USA 79, 2554–2558 (1982).

R. McEliece *et al.*, "The Capacity of the Hopfield Associative Memory," IEEE Trans. Info. Theory IT-33, 461–482 (1987).

B. Telfer and D. Casasent, "Ho-Kashyap Associative Processors," Proc. Soc. Photo-Opt. Instrum. Eng. 1005, 77–87 (1988).

5. T. Kohonen, *Self-Organization and Associative Memory* (Springer-Verlag, Berlin, 1987).
6. G. Stiles and D.-L. Denq, "On the Effect of Noise on the Moore-Penrose Generalized Inverse Associative Memory," IEEE Trans. Pat. Anal. and Mach. Int. PAMI-7, 358–360 (1985).
7. K. Murakami and T. Aibara, "An Improvement on the Moore-Penrose Generalized Inverse Associative Memory," IEEE Trans. Syst. Man and Cybern. SMC-17, 699–707 (1987).
8. D. Casasent and B. Telfer, "Key and Recollection Vector Effects on Heteroassociative Memory Performance," Appl. Opt. 28, 272–283 (1989).
9. Y.-C. Ho and R. Kashyap, "An Algorithm for Linear Inequalities and Its Applications," IEEE Trans. Electron. Comput. EC-14, 683–688 (1965).
10. T. Cover, "Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition," IEEE Trans. Electron. Comput. EC-14, 326–334 (1965).
11. M. Hassoun, "Two-Level Neural Network for Deterministic Logic Processing," Proc. Soc. Photo-Opt. Instrum. Eng. 881, 258–264 (1988).
12. M. Hassoun and D. Clark, "An Adaptive Attentive Learning Algorithm for Single-Layer Neural Networks," IEEE Int. Conf. Neural Networks I 431–440 (1988).
13. M. Hassoun, "A High-Performance Associative Neural Memory (ANM) for Pattern Recognition," Proc. Soc. Photo-Opt. Instrum. Eng. 956 (1988).
14. M. Hassoun and A. Youssef, "High Performance Recording Algorithm for Hopfield Model Associative Memories," Opt. Eng. 28, 46–54 (1989).
15. M. H. Hassoun, "Adaptive Dynamic Heteroassociative Neural Memories for Pattern Classification," Proc. Soc. Photo-Opt. Instrum. Eng. 1053, 75–83 (1989).
16. A. M. Youssef and M. H. Hassoun, "Dynamic Autoassociative Neural Memory Performance vs. Capacity," Proc. Soc. Photo-Opt. Instrum. Eng. 1053, 52–59 (1989).
17. B. Kosko, "Bidirectional Associative Memories," IEEE Trans. Syst. Man and Cybern. SMC-18, 49–60 (1988).
18. B. L. Montgomery and B. V. K. Vijaya Kumar, "An Evaluation of the Use of the Hopfield Neural Network Model as a Nearest-Neighbor Algorithm," Appl. Opt. 25, 3759–3766 (1986).
19. R. P. Lippmann, "An Introduction to Computing with Neural Nets," IEEE ASSP Mag. 4, 4–22 (1987).
20. A. D. Fisher, W. L. Lippincott, and J. W. Lee, "Optical Implementations of Associative Networks with Versatile Adaptive Learning Capabilities," Appl. Opt. 26, 5039–5054 (1987).
21. G. Strang, *Linear Algebra and Its Applications* (Harcourt, Brace, Jovanovich, New York, 1980).
22. R. Duda and P. Hart, *Pattern Classification and Scene Analysis* (Wiley, New York, 1973).
23. Y.-C. Ho and R. Kashyap, "A Class of Iterative Procedures for Linear Inequalities," J. SIAM Control 4, 112–115 (1966).
24. J. Goodman *et al.*, "Parallel Incoherent Optical Vector-Matrix Multiplier," Technical Report L-723-1, BMD (1979).
25. D. Casasent, J. Jackson, and C. Neuman, "Frequency-Multiplexed and Pipelined Iterative Optical Systolic Array Processors," Appl. Opt. 22, 115–124 (1983).
26. D. Casasent and J. Jackson, "Space and Frequency-Multiplexed Optical Linear Algebra Processor: Fabrication and Initial Tests," Appl. Opt. 25, 2258–2263 (1986).

27. K. Wagner and D. Psaltis, "A Space Integrating Acousto-Optic Matrix-Matrix Multiplier," Opt. Commun. 52, 173–177 (1984).
28. R. Winder, "Bounds on Threshold Gate Realizability," IEEE Trans. Electron. Comput. EC-12, 561–564 (1963).
29. C. Giles and T. Maxwell, "Learning, Invariance, and Generalization in High-Order Neural Networks," Appl. Opt. 26, 4972–4978 (1987).
30. D. Psaltis, C. Park, and J. Hong, "Higher Order Associative Memories and Their Optical Implementations," Neural Networks 1, 149–163 (1988).
31. Y. Kosugi and Y. Naito, "An Associative Memory as a Model for the Cerebellar Cortex," IEEE Trans. Syst. Man Cybern. SMC-7, 95–98 (1977).
32. H. Kasden, "Industrial Applications of Diffraction Pattern Sampling," Opt. Eng. 18, 496–503 (1979).

Proc. SPIE
April 1992, Orlando
Vol. 1709 (Paper 1709-03)

"LARGE CAPACITY NEURAL NETS FOR SCENE ANALYSIS"

DAVID CASASENT
Carnegie Mellon University
Department of Electrical and Computer Engineering
Center for Excellence in Optical Data Processing
Pittsburgh, PA 15213

ABSTRACT

We consider the classification of multiple objects in a scene with distortion and clutter present. Our opinions on the role for neural nets (NNs) in this application and the different properties that NNs must have to address this problem are advanced. A hierarchical/inference approach is suggested using correlation NNs for low-level operations and new classifier NNs with higher-order decision surfaces for the final decision NNs. Our concern is NN capacity and performance (in noise). Our capacity guidelines advanced concern the number of neurons, use of analog neurons, Ho-Kashyap (HK) NNs, and two new NNs with higher-order decision surfaces. Our noise performance guidelines advanced concern the number of neuron layers, hidden-layer neuron encoding, and robust HK NNs.

1. INTRODUCTION

For the demanding problem considered, we feel that even NN solutions should use a hierarchical/inference approach. The levels in such a system [1] are shown in Figure 1 and discussed briefly in Figure 2. Subsequent sections address the role for NNs in each level and the different NN properties required in each level (hence our use of a hierarchical approach, as is used in ATR [2]). Extensive use is made of prior work since much of it does not seem to generally be appreciated, possibly due to the vast quantity of NN literature. Carnegie Mellon work is emphasized, since we are most familiar with it and since it has addressed the problem we consider.

2. CORRELATION NNs

These are used for the detection, enhancement and feature extraction levels in Figure 1. The detection NN is the lowest-level processor. It operates on the entire scene and its function is to locate candidate regions of interest (ROIs). Since this level requires handling object distortions, multiple objects, and clutter, we do not attempt discrimination (identification) initially. With multiple objects present, a parallel solution

requires shift-invariance (SI). With clutter present, a parallel solution requires a large spatial set of weights (space bandwidth product) and hence a correlator (for processing gain). Figure 3 shows the standard 2-layer NN with shift invariant Fourier transform (FT) interconnections. The weights at P_2 are applied to every P_1neuron in parallel and the P_3 outputs are the weighted sum of the product of the weights and each input region with a P_3 nonlinearity (threshold etc.) applied. This is a NN version of the standard correlator (Figure 4) and hence we refer to it as a correlation NN. Alternative NNs use N_1^4 interconnections (there are N_1 input P_1 neurons) to achieve [3] SI. With $N_1 \approx 10^6$ iconic neurons, this is excessive and free space SI FT interconnections and FT weights clearly appear preferable to N_1^4 interconnections [4] when SI is required.

2.1 DETECTION NN

We use hit-miss (H-M) weights (filters) or rank-order filter techniques applied to the input scene and its complement, threshold the two P_3 outputs, and intersect the H and M P_3 outputs to achieve detection. Figure 5 shows an example of the detection of 7 input ROIs with this technique. It handles hot, cold and bimodal objects as seen. The initial algorithm has been detailed [5], demonstrated [6] and its advanced variations [7] performed very successfully in a wide range of strong background clutter.

2.2 ENHANCEMENT NN

Prior to attempting classification of the object in each ROI, it is generally advisable to enhance each ROI. This involves noise removal, filling in holes on the object, edge enhancement, etc. Since the location of the object in each ROI is not known, enhancement requires SI and thus we use our correlation NN (Figure 3). Now N_1 is smaller (only the ROI pixels are input to P_1). The P_2 weights used are now morphologically inspired and are simple uniform structuring element (SE) filters (disks etc.). The spatial size of each SE weight function determines the size of the holes filled in on the object and the size of noise regions omitted. The P_3 neuron thresholds used define the operation performed: a low threshold yields a dilation and a high threshold yields an erosion. The difference between dilation and erosion images yields an edge enhanced image. Figure 6 shows examples of these operations. Some similar operations can also be achieved using NN retina chips [8] etc. Their correlation NN realization and the many operations possible (besides those shown in Figure 6) are detailed elsewhere [9].

2.3 FEATURE EXTRACTION NN

Prior to classifying an object, features are generally extracted to describe each ROI. Since the location (or even the presence) of an object in each ROI is not known, SI is again required. We thus prefer to again use the correlation NN (Figure 3) for feature extraction. In this case, the NN weights at P_2 are chosen using computer generated hologram (CGH) techniques, which can achieve a larger number of different feature spaces at the P_3 neuron outputs [10]. Many NNs have been described that can calculate features such as edges, moments, Hough transforms, etc. However, we see no reason to use such NNs versus standard correlator or VLSI chips for these purposes. NNs can also conceptually

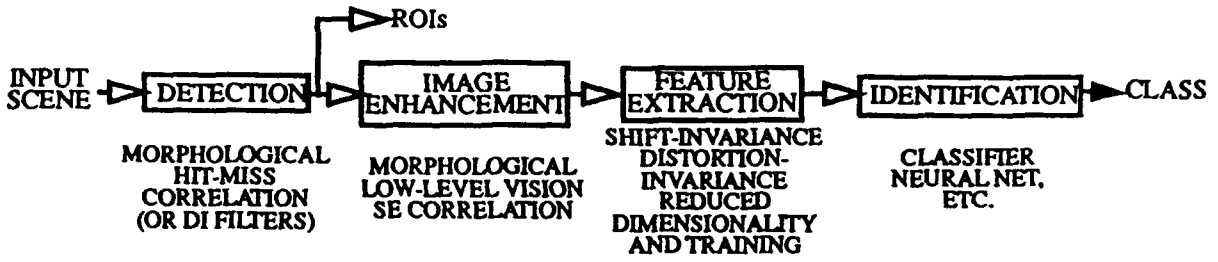


FIGURE 1: Hierarchical inference levels of scene analysis.

1) DETECTION
   MULTIPLE OBJECTS, SHIFT-INVARIANT WEIGHTS
   LOCATE REGIONS OF INTEREST (ROIs)

2) IMAGE PROCESSING (ENHANCEMENT)
   REDUCE NOISE, FILL IN HOLES, EDGE DETECTION

3) FEATURE EXTRACTION

4) IDENTIFICATION
   DETERMINE CLASS OF OBJECT IN EACH ROI
   HIGHER ORDER MORE COMPLEX NN DECISION SURFACES

FIGURE 2: Remarks on levels in Figure 1.

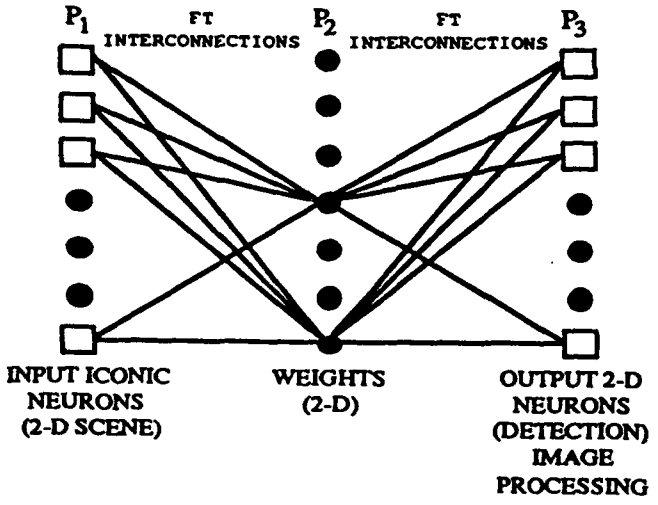


FIGURE 3: Shift invariant correlation NN.

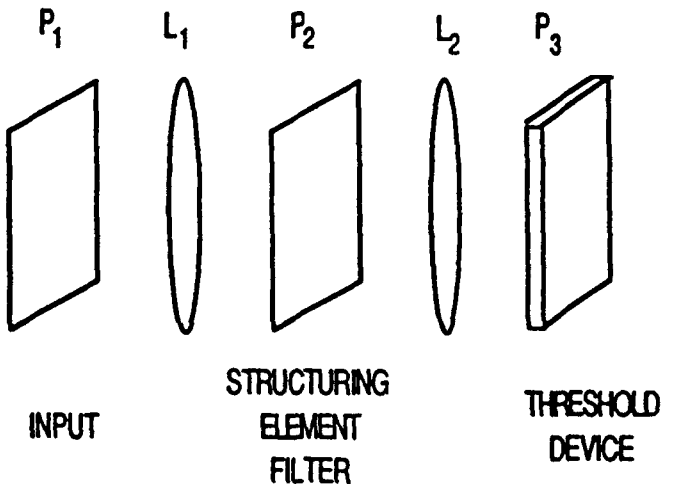
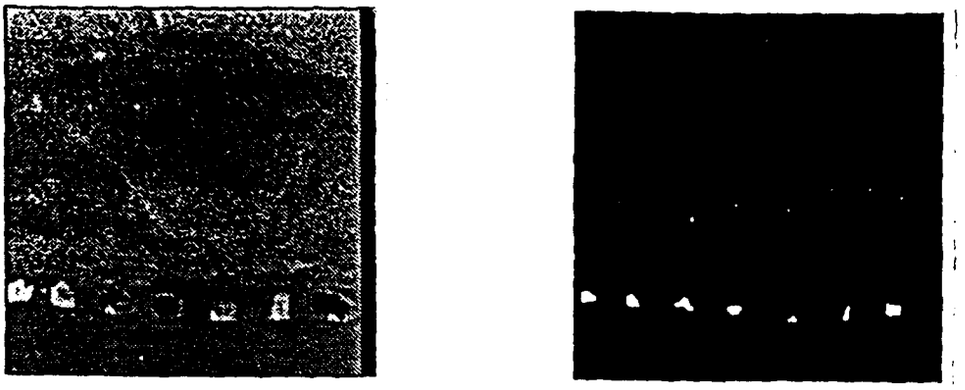
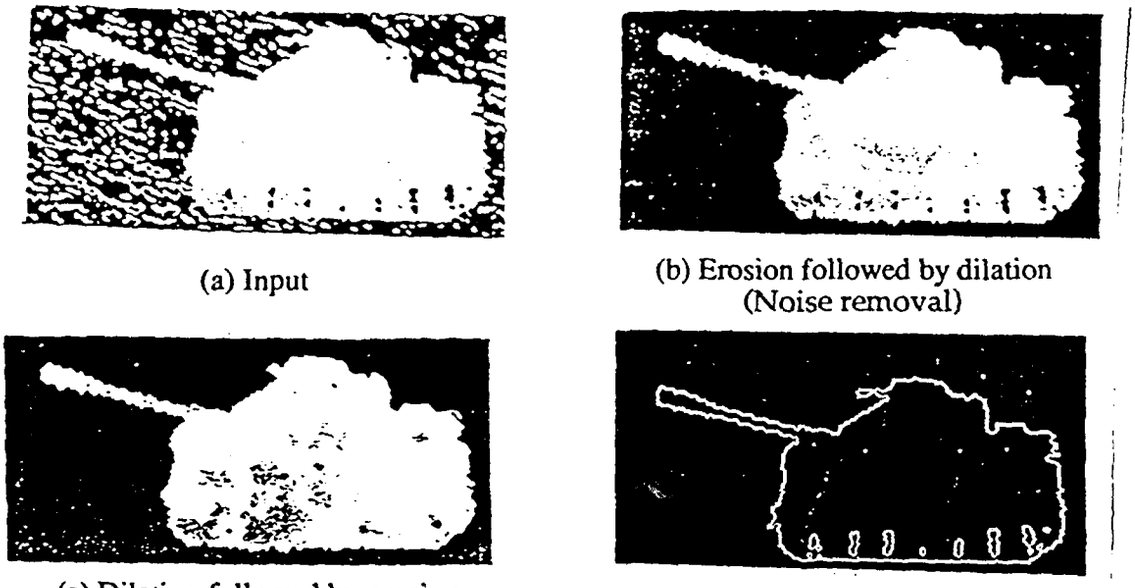


FIGURE 4: Standard correlator (optical).



(a) Input (b) Output

FIGURE 5: Detection NN example results.



(a) Input

(b) Erosion followed by dilation (Noise removal)

(c) Dilation followed by erosion (Fill in holes)

(d) Edge enhancement

FIGURE 6: Image enhancement NN example.

be used to compute a set of best features (with no *a priori* choice of the feature space). No ideal NN for this has yet emerged. Candidate solutions such as the Neocognitron [11] use an excessive number of NN layers and other solutions such as the ART [12] require complex individual neuron elements (with a parallel array of such elements for each input pixel).

2.4 UNIFIED MULTIFUNCTIONAL NN

An attractive aspect of our NN approach to the first 3 processing levels noted in Figure 1 is that the same correlation NN architecture is used with different P_1 input neurons (the full scene or an ROI) and different weights at P_2 (H-M, SE, CGH choices).

3. NN CAPACITY

The classifier NN we consider is the standard 3-layer NN in Figure 7 (i.e. with one hidden layer of neurons at P_2). The notation we use considers N_1input neurons (features) at P_1, N_2 hidden layer neurons at P_2 and N_3output neurons at P_3 (N_3 is the number of object classes, although P_3 encoding can represent more than N_3 classes). We denote the number of training set images (vectors) by N_T.

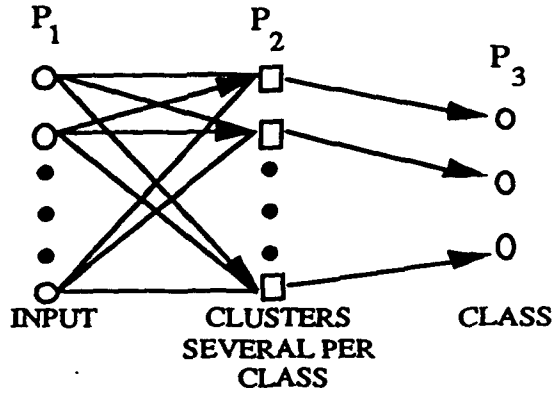


FIGURE 7: Basic 3-layer classification NN considered.

3.1 NUMBER OF LAYERS

A 3-layer NN can produce any decision surface [13,14] and hence is used. The proof of this requires more N_2 neurons to better approximate higher-order surfaces. The new higher-order NNs we consider (Section 4) allow higher-order surfaces and exact (not approximate) higher-order surfaces with few N_2 neurons. When noise is considered, one can show [15] that performance degrades with more neuron layers since errors propagate. We also consider only 3-layer NNs since we have not yet found a good algorithm (without ad hoc parameters) for training NNs with more than one hidden layer. If significant training is performed, 4-layer NNs may be able to train out noise propagation effects. However, 3-layer NNs are clearly preferable if they perform well (as ours do), as they are more well defined and less ad hoc. Work on mapping decision trees into NNs [16] is not considered as it leads to many neural layers and often binary neurons, it scales poorly with increased size and is contrary to parallel and NN concepts. Decision trees also classically use one cluster per class (while we find that several N_2 neurons per class is preferable).

3.2 ANALOG NEURONS AND WEIGHTS

High capacity requires analog input P_1 neurons and analog weights from P_1-P_2. Polarization methods [17] to achieve optical bipolar data (in optical NNs) are restricted to only binary neurons and weights and require hard clipped output neurons. Thus,

they are not of use. Since image pixels and features are analog, the analog P_1 neuron requirements are essential.

3.3 NUMBER OF INPUT N_1 NEURONS

As N_1 increases, so does capacity. However, numerical stability is now of concern [18] (i.e. the defined problem is ill conditioned). This also translates into increased required numeric accuracy in the P_1 neurons and the P_1-P_2 weights. A larger N_1 also requires a larger training set (N_T) and the curse of dimensionality [19] then enters. With many N_1 features, some features will be of little use (small variance for the different classes) and other features will have large variances. Thus, a larger N_1 is good for capacity, but introduces considerable practical problems (noise, instability, accuracy requirements, a large N_T, many local minima). Our hierarchical approach relieves these problems, but one should still restrict N_1 (our use of feature space P_1 neurons addresses this). Numerical accuracy requirements clearly increase with N_1 (they also increase with N_2, but the affect seems much less).

3.4 NUMBER OF HIDDEN LAYER NEURONS N_2

3.4.1 Approaches

No general solution yet exists to this. However, it seems well worthwhile to advance remarks on various approaches to determining N_2. A number of papers have established a bound of $N_2 = N_T - 1$, but this assigns one neuron to nearly each of the N_T training image and N_2 is too large to be practical. Other derivations make unrealistic assumptions not valid for general data that contains distorted images, etc. Neural nets that perform piecewise linear input-to-output mapping generally use a large $N_2 \approx N_T - 1$ and thus memorize and perform a desired functional mapping (this is not the problem addressed in our classifier NNs). Many methods simply increase or decrease N_2 until adequate performance is obtained. We desire non ad hoc methods and a good initial N_2 choice (else training time is excessive, optimization is not necessarily obtained and comparisons are not easily possible). Techniques [20] that select the number of N_2 neurons per class based upon the *a priori* probability of each class occurring did not perform well (we attribute this to the fact that the number of N_2 neurons per class should be based on how disjoint a class is and how similar two classes are). Methods which use linear algebra [21] and covariance [22] techniques to calculate subspaces etc. cannot be applied to large N_1 and N_T cases [21,22] and often [21] apply only to binary neurons. Pruning/removing weights does not [23] always yield reliable results and causes data to become linearly inseparable; thus we ignore such methods and note that decreasing N_2 is not yet easy and adding additional layers [23] when reducing N_2 causes other problems.

3.4.2 General N_2 Remarks

We now advance several obvious (but not quantitative) general remarks on N_2. Small N_2will not allow the problem to be solved. Excessive N_2 results in memorizing the

training data and generalization (good test data results) does not occur (and local minima may arise). With N_2 neurons, one has N_2 regions in decision space and can form a maximum of $(N_2)(N_2\text{-}1)$ decision boundaries (lines), one between each N_2 pair. In practice, the number of decision boundaries is much less as many are not of use (such as ones between two N_2 neurons in the same class). No clear relationship between N_2 and the number of classes N_3 can generally be obtained. Some insight into how N_2 relates to decision surfaces [22] exists, but no decision emerges in general. Clearly capacity increases as N_2 increases and N_2 increases as N_1 and N_3 increase. N_2 relates to capacity and decision surfaces. Our new higher-order decision surface NNs (Section 4) address this problem. They provide improved P_C with fewer N_2 neurons and thus are preferable. These methods are best seen by attention to linear discriminant functions and how NNs produce decision surfaces. Section 4 addresses these issues.

3.4.3 Preferable Approach (Prototypes)

The technique we use in the Adaptive Clustering NN (ACNN) [4] to select N_2 uses prototypes obtained from the training set. This concept has long been used [24,25] to extend linear classifiers to piecewise linear ones. The linear vector quantizer [26] (LVQ) uses the PDF of each class and the data to select N_2. This results in a larger N_2 than in the ACNN. In our NN, we are concerned with decision surfaces not with modeling data PDFs (specifically, our concern is with the boundaries between data clusters not the means of such clusters). We feel it is important to not pick initial prototypes at random or uniformly distributed (as LVQ does) as this yields larger N_2. We also feel that use of more than one prototype per class is needed, but these should not be arbitrarily used. Rather, they should be used when class data is disjoint and discrimination between two similar classes is needed. This is somewhat related to K-nearest-neighbor (KNN) classifiers (but our motivation is different and no NN as yet performs KNN - rather many KNN designs are presented as NNs).

Our clustering technique [4] to select N_2 allows a rapid analysis of the training set followed by a second selection process to pick the best prototypes (more than one per class). These are chosen to both represent the data, but primarily to discriminate classes (i.e. we consider the boundaries between clusters rather than the mean of each cluster). This choice (rather than random prototypes) and the use of more than one per class distinguishes our method from others. We use these prototypes to set initial P_1-P_2 weights and then we adapt them. Our initial weights and adaptation of them differ from other prototype methods. We include an additional P_1 neuron with input 1 and weights $-0.5\sum w_{ij}^2$. As the NN algorithm adapts, this constraint on the added N_1 neuron is not enforced (in LVQ it is still enforced). In LVQ, this forces decision boundaries to lie on the perpendicular bisector (midpoint) between prototypes. In the ACNN, the decision surface need not lie at the midpoint. Thus, the ACNN yields more flexibility and better P_C results with fewer N_2 neurons.

3.4.4 Prototype Extensions

Care must be taken to not select N_2 too large. We generally achieve a correct classification of only $P_C \approx 50\%$ with the initial N_2 prototypes. If a small N_2 yields high

P_C, then a NN may not be needed. However, test results may be poor with this N_2choice. One can check test set performance every few hundred adaptations to verify this and lower N_2 and restart if needed. If N_2 is too small, the NN adaptations provide this information and we then increase N_2. To remove dependence on the order in which the training data is presented, we train in batch mode. We avoid outliers by our N_2 selection. If the input image training data contains many artifacts, it is essential to not overtrain in NN adaptations. If noise is present in the training data, then distributions can help avoid noise effects. Training on noisy data is now needed. Use of PDFs can be of use here, but proper selection of N_2 yields similar results with a preferable algorithm. In general, one should not add additional N_2 neurons to handle several training images that are a small percentage of a class or of N_T.

With a good initial N_2 estimate (as above), we can easily increase N_2. The new N_2neuron added would be one to handle the class or discrimination between classes needing assistance. Our clustering provides data on the next best such prototype. Our N_2 selection method is preferable to N_2 choices based on the PDF of each class, since we consider discriminating classes not modeling of a class. (i.e. attention to class boundaries not class means) and we can far more easily choose a new N_2 neuron and its weights. The new prototype sets these weights and they are not random as others [27] use; all other prior weights are kept unchanged. We know, from the NN results, the classes needing help and from clustering we know the prototypes to add.

3.5 HO-KASHYAP (HK) NNs

For the best NN storage, we can and have [28,29] quantified the best algorithm to select the P_1-P_2 weights. For inputs in general position, this is the HK NN algorithm. It handles dependent inputs and yields storage of $N_3 = 2N_1$ classes. This is theoretically the maximum (using WTA P_2neurons). We have shown [28] that the best results with input noise occur for the robust HK algorithm. We have also shown that this yields adequate analog input neuron accuracy and P_1-P_2 weights when limited analog accuracy is present. This is relevant to optical and analog VLSI NNs. The noise control parameter σ_{syn} in the HK algorithm is selected to optimize such noise conditions and accuracy limitations. We found that L-max P_2neuron encoding yields the best P_C in noise etc. These results do not seem to be widely known. Our recent [30] results show that with this algorithm we can also train out internal NN accuracy and non-ideal device effects. We have consistently achieved better P_C results with this algorithm than with an infinite accuracy NN quantized (after the fact) to lower input and weight accuracy.

3.6 PERFORMANCE

The performance we consider is the percent correct classification P_C of the input data. To be meaningful, one must obtain P_C > 90-95% (50-70% performance is not of general use) and this must be obtained with input noise and non-training test data with $N_3 > N_1$ storage. It is also crucial to be able to modify the same NN to maximize P_C and/or minimize P_e (probability of error). We will show how this is possible (Section 4). To improve P_C and P_e performance, higher-order decision surfaces are needed. Section 4

addresses new NNs to achieve this (we achieve exact, not approximate, surfaces without the need for large N_2). Another issue of concern in NNs is training time. Our use of conjugate gradient methods [4] is much faster (by a factor of 100 to 1000) than standard gradient descent or delta rule neuron update algorithms. This result does not seem to be widely known.

4. NN DECISION SURFACES

4.1 PIECEWISE LINEAR DECISION SURFACES

We produce piecewise linear discriminant surfaces by using initial P_1-P_2 weights that are linear discriminant functions (LDFs) set by N_2 exemplars and using NN techniques to adapt and combine these. Figure 8 shows [4] an example for $N_1 = 2$ input neurons and $N_3 = 3$ classes using $N_2 = 6$ hidden layer neurons. As seen, the 3 classes of data are separated and nonlinear surfaces are required to achieve this. We include an additional input N_1 neuron whose input is 1 and whose weights are related to the sum of the squares of the other weights. This allows the threshold for each P_2 neuron to be separately adapted. This allows each line in Figure 8 to be shifted (i.e. it need not pass through the origin) and this (together with our non ad hoc N_2 selection) avoids local minima. This also insures nearest neighbor convergence (the closest P_2 prototype neuron is the most active one). We note that a nonlinearity at neuron layer P_2 is essential otherwise a linear NN results, which is merely a standard pattern recognition LDF and not a true NN. We now discuss two other techniques using only 1 or 2 additional input neurons that allow higher-order decision surfaces. These methods are preferable to others that use an excessive number of interconnections N_1^4 to provide higher-order weights.

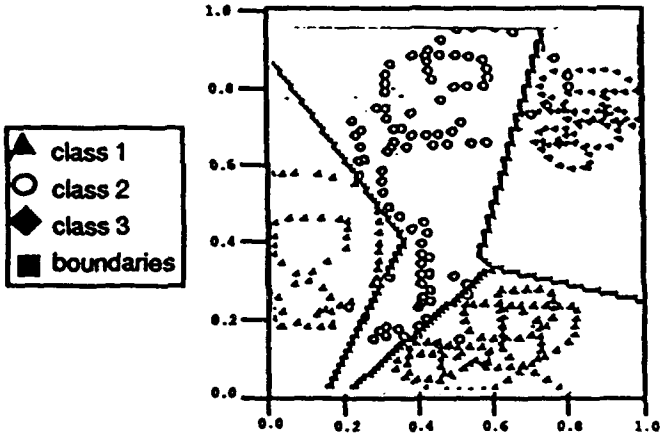


FIGURE 8: Piecewise nonlinear decision surfaces.

4.2 PIECEWISE HYPERSPHERICAL NN [31]

This concept is best shown for the case of $N_1' = 2$ two input neurons with an input described by $\underline{x}' = [x_1 x_2]^T$. We add 2 neurons and use $N_1 = 4$ neurons

$\underline{x} = [1\ (x_1^2+x_2^2)\ x_1\ x_2]^T$, and we denote the weights to hidden layer neuron n by $\underline{w}_n = [w_1 w_2 w_3 w_4]^T$. The decision boundary for each P_2 neuron is $\underline{x}^T\underline{w}_n = 0$ or $w_1^2 + w_2(x_1^2+x_2^2) + w_3x_1 + w_4x_2 = 0$. As seen this describes a circle. It also describes a line (if $w_2 = 0$). If $N_1' > 3$, it is a hypersphere. Each P_2 neuron can now describe a hypersphere and the decision surfaces are piecewise hyperspheres. To design this NN, we add several N_2 neurons per class (as needed) to separate one class from the others. To use this neural net, we simply look at the signs of the P_2 neurons. If each neuron is > 0 (< 0), the sign denotes if the corresponding input is inside (outside) the sphere for that N_2 neuron. Note that each hypersphere is easily designed in the new N_1 space as an LDF but when used in the original N_1' space is a hypersphere.

Figure 9 shows an example of the decision surfaces produced. We consider 3 classes of data, $N_1 = 4$ neurons and $N_2 = 6$ hidden layer neurons. This NN produces $N_2 = 6$ circles (C_1-C_3 for class 1 etc.) that define 12 regions of space (R_1-R_3 for class 1, etc., and R_{10}-R_{12} in which no test data lies). The signs of the $N_2 = 6$ neurons specify the location of an input in R_1-R_{12} as inside/outside each hypersphere. Figure 10 shows the 3 class boundaries defined by piecewise hyperspheres. Note that an input lying in R_{10}-R_{12} can be assigned no decision or classified as a reject class. If a WTA is used at P_2, then hard decisions (no reject class) result and the decision boundaries in Figure 11 occur (with no reject class). These Figure 11 surfaces yield much better P_C and the surfaces in Figure 10 yield much better P_e as we will show (Section 4.4).

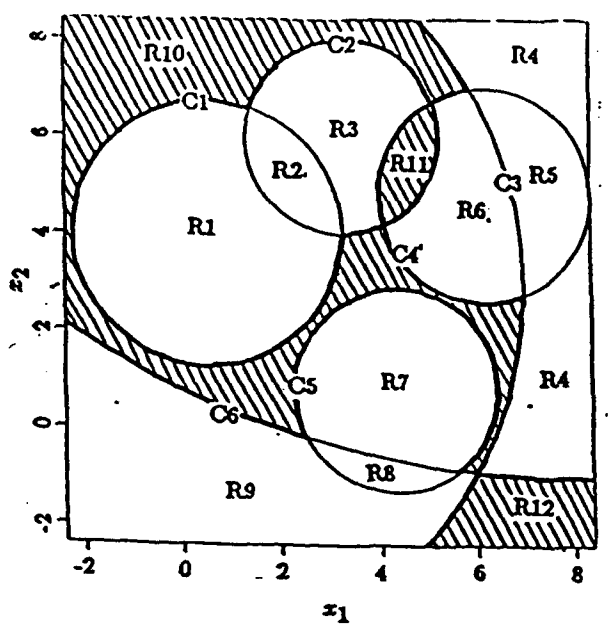


FIGURE 9: Twelve regions R defined by 3 circles in the hyperspherical NN.

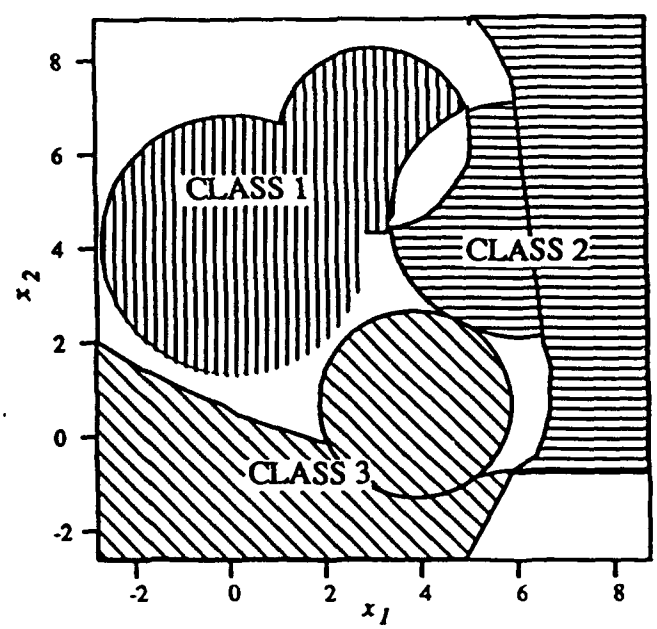


FIGURE 10: Three class (and reject class) decision regions from Figure 9.

4.3 PIECEWISE HYPERQUADRATIC NN [32]

This provides even more flexibility in the decision surfaces produced and allows better P_C and P_e. This uses one additional input neuron $\underline{x} = [1\ x_1\ x_2]^T$, complex-valued weights $\tilde{w}_{ij} = w_{ij} + jv_{ij}$ (these are easily produced optically) and nonlinear intensity detection at P_2 (as optics provides). For $N_1 = 3$ input neurons and only $N_2 = 2$ hidden layer neurons, the $N_2 = 2$ outputs are

$$f_1(\underline{x}) = \left\| \tilde{w}_{11}x_1 + \tilde{w}_{12}x_2 + \tilde{w}_{13} \right\|^2$$

$$f_2(\underline{x}) = \left\| \tilde{w}_{21}x_1 + \tilde{w}_{22}x_2 + \tilde{w}_{23} \right\|^2$$

and the decision surface produced, $f_1(\underline{x}) = f_2(\underline{x})$, is

$$ax_1^2 + bx_2^2 + cx_1x_2 + dx_1 + ex_2 + f = 0$$

As seen, each pair of P_2 neurons in this NN can produce any quadratic surface required. A line results (if $a = b = c = 0$), a circle or an ellipse ($a \neq b$) results (if $c = d = e = 0$), etc. These surfaces are exact (not approximate as in BP etc.). As before, they are easily calculated as LDFs in the new space and are hyperquadratic (if $N_1 > 2$) in the original space (in which they are used). Figure 12 shows some of the various surfaces possible (the NN algorithm and the data determine the complexity required in each surface).

4.4 INITIAL RESULTS

We show results for a real multiclass identification problem with severe distortions present. We consider 3 classes of aircraft with several (±80°) pitch and roll distortions. Figure 13 shows several distorted inputs for one object class (DC10). We used $N_1 = 34$ neurons (32 wedge Fourier samples and 2 additional neurons) and $N_2 = 24$ hidden layer neurons. The decision surfaces produced were piecewise nonlinear combinations of 24 different hyperspheres ($N_2 = 24$). The training set used consisted of 3267 distorted inputs. The test set was 3072 other distorted inputs not present in the training set. The test results (Table 1) are excellent. They show the flexibility of the NN to optimize P_C or P_e for a given problem. If we desire large P_C, we use maximum selection (WTA neurons) and achieve $P_C = 98.9\%$. If we desire low P_e, we use thresholded ($\gtrless 0$) neurons and achieve $P_e = 0.4\%$. These results are most impressive considering the severe distortions present and the few N_1 and N_2 neurons used.

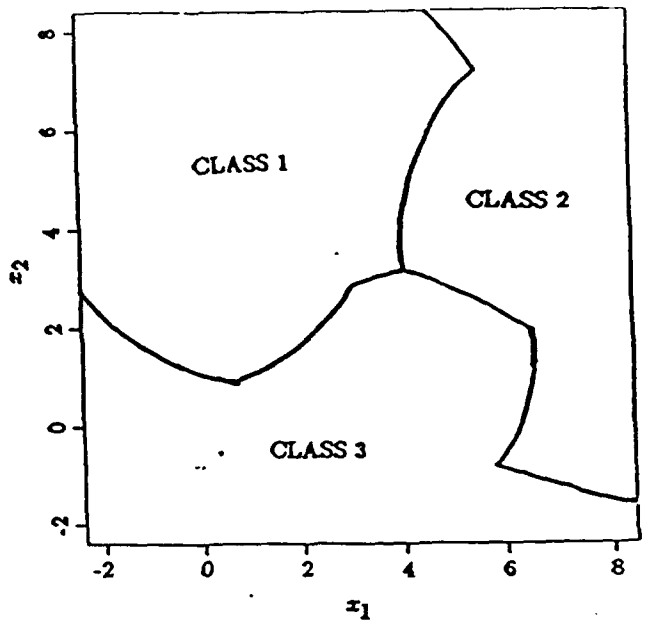


FIGURE 11: Class boundaries produced by WTA neurons in Figure 9.

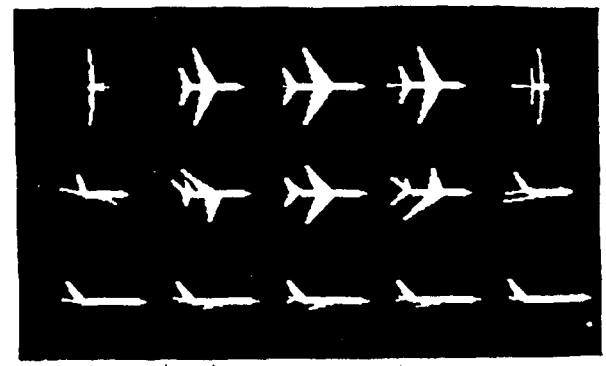


FIGURE 13: Representative DC-10 distorted input images.

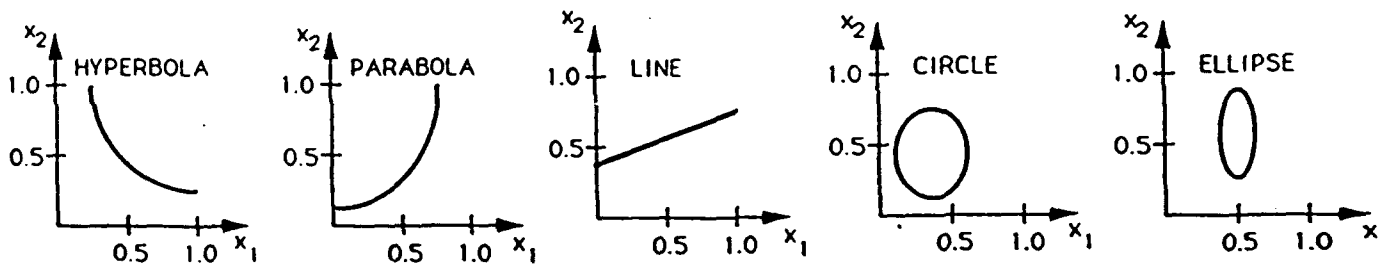


FIGURE 12: Decision surfaces produced on the hyperquadratic NN.

	THRESHOLD			MAX SELECTION	
	$P_{C(\%)}$	Reject (%)	$P_e(\%)$	$P_C(\%)$	$P_e(\%)$
Training Set	93.9	5.7	0.4	98.9	1.1
Test Set	92.5	6.8	0.7	98.1	1.9

Table 1: Test results of the hyperspherical NNs for high P_C (max selection WTA neurons) and low P_e (thresholded ($\gtrless$ 0) neurons.

For completeness, we review some of our HK NN results [28]. The storage N_3/N_1 = (number of classes)/(number of input neurons) for which $P_C > 95\%$ can be quantified for various associative processors. The Hopfield NN (using a correlation matrix) yields poor $N_3 = 0.12N_1$ storage ($N_3 << N_1$), the pseudoinverse solution (using standard linear algebra not NNs) is much better ($N_3 = 1.04N_1$ or $N_3 \sim N_1$) and the HK NN is best ($N_3 = 1.52N_1$). When input noise is present ($\sigma_n = 0.1$), all NNs degrade. We ignore the Hopfield NN since its performance is too poor (and no variations of it can sufficiently improve results). For $P_C > 95\%$, the standard HK NN can now only store $N_3 = 0.75N_1$ (this is still better than the linear algebra pseudoinverse solution), while the robust HK NN is better ($N_3 = 0.8N_1$). However, better results (equal to the noise free ones) with $N_3 > N_1$ are obtained with L-max (L = 2) encoding where $N_3 = 1.5N_1$ (no loss) is obtained. Our analog accuracy tests are also most impressive and show that analog accuracy NNs can achieve better performance than infinite accuracy NNs when trained on limited accuracy data using the robust HK NN. In the HK NN synthesis, we now use $\sigma_n = 2^{-B}/12^{1/2}$ for a B-bit accuracy NN. With infinite accuracy, we obtained $P_C = 82.3\%$ and the same performance when the inputs etc. to this NN were quantized to 8 bits. However, our robust HK NN with σ_{syn} and with training on 8 bit data yielded much better (90.6%) results.

TEST RESULTS TEST SET 1 = ±50° DISTORTIONS (N_T = 882) TEST SET 2 = ±60° DISTORTIONS (N_T = 1250)				
NEURAL NET USED	TEST-SET 1		TEST-SET 2	
	TRAIN	TEST	TRAIN	TEST
STD HK	93.2	92.3	82.3	88.5
ROBUST HK	95.1	95.4	90.6	91.2

Table 2: HK NN tests.

Very significant multiclass (3 different aircraft) recognition with severely distorted inputs (±50° and ±60° distortions in pitch and roll) were obtained. The NN used $N_1 = 33$ input neurons and only $N_2 = N_3 = 2$ neurons. Table 2 shows the test results obtained. As seen, the robust HK NN provided 2-8% better results. These very impressive results are highlighted in Table 3.

3-D DISTORTION-INVARIANT MULTICLASS IDENTIFICATION 33 INPUT NEURONS, 2 HIDDEN LAYER NEURONS RECOGNIZE $P_C' > 95\%$ OF OVER N_3= 2000 INPUTS > $70N_1$ IN NOISE ($\sigma_n = 0.1$) WITH 6 BIT (1%) ACCURACY NN

Table 3: Very impressive accuracy and noise Table 2 results summarized.

REFERENCES

1) D. Casasent, "An Optical Correlator, Feature Extractor, Neural Net System", *Optical Engineering*, May 1992.

2) B. Bhanu, "Automatic Target Recognition: State of the Art Survey", *IEEE Trans. Aerosp. and Elec. Syst.*, AES-22, pp. 364-379, July 1986.

3) C.L. Giles, R.D. Griffen and T. Maxwell, "Encoding Geometric Invariances in Higher-Order Neural Networks", *Neural Information Processing Systems*, D. Anderson, ed., pp. 301-309, 1988, Denver, Colorado, AIP.

4) D. Casasent and E. Barnard, "Adaptive Clustering Optical Neural Net", *Applied Optics*, Vol. 29, pp. 2603-2615, 10 June 1990.

5) D. Casasent, R. Schaefer and R. Sturgill, "Optical Hit-Miss Morphological Transform", *Applied Optics*, 1992.

6) D. Casasent, R. Schaefer and J. Kokaj, "Morphological Processing to reduce Shading and Illumination Effects", *Proc. SPIE*, Vol. 1385, pp. 152-164, November 1990.

7) D. Casasent and R. Schaefer, "Optical Gray Scale Morphology for Target Detection", *Proc. SPIE*, Vol. 1568, pp. 313-326, July 1991.

8) C.A. Mead and M.A. Mahowald, "A Silicon Model of Early Visual Processing", *Neural Networks*, Vol. 1(1), pp. 91-97, 1988.

9) D. Casasent, "Optical Morphological Processors", *Proc. SPIE, Vol.* 1350, pp. 380-394, July 1990.

10) D. Casasent, "Computer Generated Holograms in Pattern Recognition: A Review", *Optical Engineering*, Vol. 24, pp. 724-730, September/October 1985.

11) K. Fukushima, "Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by a Shift in Position", *Biological Cybernetics*, Vol. 36, pp. 193-202, 1980.

12) S. Grossberg and E. Mingolla, "Neural Dynamics of Form Perception: Boundary Completion, Illusory Figures, and Neon Color Spreading", *Psychological Review*, Vol. 92(2), pp. 173-211, 1985.

13) R.P. Lippmann, "An Introduction to Computing with Neural Nets", *IEEE ASSP Magazine*, pp. 4-22, April 1987.

14) G. Gibson and C. Cowan, "On the Decision Regions of Multilayer Perceptrons", *IEEE Proceedings*, Vol. 78(10), pp. 1590-1594, October 1990.

15) D. Lovell, P. Bartlett and T. Downs, "Error and Variance Bounds on Sigmoidal Neurons with Weight and Input Errors", submitted to *Electroncis Letters*.

16) K.J. Cios and N. Liu, "A Machine Learning Method for Generation of a Neural Network Architecture: A Continuous ID3 Algorithm", *IEEE Trans. on Neural Networks*, **Vol. 3**(2), pp. 280-291, March 1992.

17) C. Gomes, H. Sekine, T. Yamazaki and S. Kobayashi, "Bipolar Optical Neural Networks Using Ferrorelectronic Liquid Crystal Devices", *Neural Networks*, **Vol. 5**, pp. 167-177, 1992.

18) O.M. Omidvar and C.L. Wilson, "Massively Parallel Implementation of Neural Network Architectures", *Proc. SPIE*, **Vol. 1452**, pp. 532-543, 1991.

19) R.O. Duda and P.E. Hart, **Pattern Analysis and Machine Intelligence**, John Wiley and Sons, New York, 1973.

20) S. Geva and J. Sitte, "Adaptive Nearest Neighbor Pattern Classification", *IEEE Trans. on Neural Networks*, **Vol. 2**(2), pp. 318-325, March 1991.

21) O. Fujita, "A Method for Designing the Internal Representation of Neural Network sand Its Application to Network Synthesis", *Neural Networks*, **Vol. 4**, pp. 827-827, 1991.

22) A.R. Webb and D. Lowe, "The Optimized Internal Representation of Multilayer Classifier Networks Performs Nonlinear Discriminant Analysis", *Neural Networks*, **Vol. 3**, pp. 367-375, 1990.

23) J. Sietsma and R. Dow, "Creating Artificial Neural Networks that Generalize", *Neural Networks*, **Vol. 4**, pp. 67-79, 1991.

24) RO. Duda and H. Fossum, "Pattern Classification by Iteratively Determined Linear and Piecewise Linear Discriminant Functions", *IEEE Trans. Elec. Comp.*, **EC-15**, pp. 220-232, 1966.

25) T. Kohonen, **Self-Organization and Associative Memory**, Springer-Verlag, New York, 1987.

26) J.A. Kangas, T.K. Kohonen and J.T. Laaksonen, "Variants of Self-Organizing Maps", *IEEE Trans. Neural Networks*, **Vol. 1**(1), 93, 1 March 1990.

27) Y. Hirose, K. Yamashita and S. Hijiya, "Back-Propagation Algorithm which Varies the Number of Hidden Units", *Neural Networks*, **Vol. 4**, pp. 61-66, 1991.

28) D. Casasent and B. Telfer, "High Capacity Pattern Recognition Associative Processors", *Neural Networks*, 1992.

29) B. Telfer and D. Casasent, "Ho-Kashyap Optical Associative Processors", *Applied Optics*, **Vol. 29**, pp. 1191-1202, 10 March 1990.

30) D. Casasent, L. Neiberg and S. Natarajan, "Techniques for High-Performance Analog Neural Networks", *Proc. SPIE*, **Vol. 1608**, pp. 400-413, November 1991.

31) B. Telfer and D. Casasent, "Minimum-Cost Ho-Kashyap Associative Processor for Piecewise-Hyperspherical Classification", IJCNN'91 (*International Joint Conference on Neural Networks*), IEEE Catalog No. 91CH30494-4, July 1991, **Vol. II**, pp. II-89-II-94, Seattle, Washington.

32) S. Natarajan and D. Casasent, "A Piecewise Quadratic Neural Network for Pattern Classification", *Proc. SPIE*, **Vol. 1608**, pp. 500-505, November 1991.

A Piecewise Quadratic Neural Network for Pattern Classification

Sanjay Natarajan and David Casasent

Center for Excellence in Optical Data Processing
Carnegie Mellon University, ECE Department
Pittsburgh, PA 15213

Abstract

A neural network pattern classifier is presented. Its decision boundaries are formed from segments of conic sections which allows it to achieve improved performance over piecewise linear neural network classifiers, such as our earlier adaptive clustering neural network (ACNN). We discuss an optical realization that uses complex-valued weights, optical intensity detectors, and an additional input neuron to achieve piecewise conic decision surfaces (rather than the piecewise linear surfaces that the ACNN produces).

1 Introduction

Neural networks (NNs) have the ability to produce arbitrarily complex decision boundaries [1] in an organized and efficient manner. This makes them very attractive for difficult multiclass classification problems. In this paper, we extend our earlier ACNN algorithm [2] (which provided piecewise linear discriminant surfaces) to more complex piecewise quadratic decision surfaces (thereby improving recognition percentage P_c).

The ACNN [2] has several attractive properties. For example, it requires few ad hoc parameters to be selected. It uses pattern recognition and linear discriminant function (LDF) techniques to select initial weights. It then uses neural network optimization techniques to refine the initial weights to produce combinations of LDFs, forming piecewise linear decision surfaces, and it converges much faster than the standard benchmark, the backpropagation training algorithm. We now improve upon the ACNN with the piecewise quadratic neural network (PQNN).

Many researchers have noted the parallelism and interconnection advantages of optical NNs. We address many new and practical issues associated with optical architectures. We employ a feature space neuron representation space to utilize a reduced number of input neurons. Optical NNs [3,4] can conceptually implement the multilayer perceptron [5] neural network architecture (which can produce complex decision surfaces), but these require advanced optical materials and devices. We consider an optical implementation of the PQNN in which the use of an extra input neuron, complex-valued weights, and intensity detectors provides piecewise conic surfaces. We use multilevel phase error diffusion [6] to produce high accuracy complex-valued interconnection weight. The use of complex-valued weights is easily possible in optics but has not been used in optical NNs. Most optical NNs also do not use the intensity (magnitude squared) nature of optical detectors for their nonlinearity advantages. The resultant PQNN thus makes use of specific advantages of optics. Because of the complex-valued weights used in the PQNN, it is directly suited for an optical implementation.

0-8194-0745-3/92/$4.0

2 Architecture

Fig. 1 shows the basic three layer NN architecture used. The $N_1 = N + 1$ input P_1 neurons are analog (the ability to handle analog data directly is another attractive feature of an optical or analog VLSI NN). The first N neurons at P_1 are a feature space description of the multiclass input data to be classified. The input to the additional (N+1)-th neuron is always equal to unity. This neuron allows the NN to adjust the center of each hyperquadratic discriminant function. Our feature space inputs are generally unipolar. If other bipolar input representation spaces are used, we use an input P_1 neuron nonlinearity to produce unipolar input neurons

$$\underline{x} = \frac{1 + \tanh(\alpha \underline{\tilde{x}})}{2} \tag{1}$$

where the monotonic sigmoid nonlinearity maintains the ordering of the feature space data. The input-to-hidden layer (P_1-to-P_2) weights are complex-valued. A winner take all (WTA) operation is applied to the N_2 hidden layer neuron outputs. The P_2 neurons are intensity sensitive with outputs

$$f_i(\underline{x}) = \| \sum_j (w_{ij} + j v_{ij}) x_j \|^2, \tag{2}$$

where j denotes the input P_1 neuron index, i is the P_2 hidden layer neuron index, and $w_{ij} + j v_{ij}$ are the complex-valued P_1-to-P_2 weights. We determine a number of prototypes from the training set of data and from these determine the number of hidden layer neurons to be used. From the locations of the prototypes (in our feature space), we select initial quadratic decision boundaries (by looking at sets of prototypes). These boundaries define initial complex-valued P_1-to-P_2 weights and hence inital probability density clusters (in feature space) for each P_2 neuron. These initial weights are then adapted in our NN training algorithm (Section 4). When training is complete, classification is performed on the test data. In testing, the most active P_2 neuron denotes the cluster to which the input P_1 data to be classified belongs (we allow more than one P_2 cluster or prototype per class). The P_2 outputs are then mapped with binary weights to the C output P_3 neurons (one per class, where there are C classes in our multiclass problem).

3 Two Dimensional Quadratic Surfaces

To show that this architecture produces quadratic decision surfaces, we consider the simplified NN architecture of Fig. 2. This shows a two class problem (two output P_3 neurons) with two hidden layer P_2 neurons and a two dimensional feature space ($2 + 1 = 3$ input P_1 neurons). The notation used is shown in Fig. 2. The P_1-to-P_2 weights are complex-valued and are denoted by

$$\tilde{w}_{ij} = w_{ij} + j v_{ij}, \tag{3}$$

where we constrain $\|\tilde{w}_{ij}\| \leq 1.0$. Practical optical weights must satisfy this constraint as they are passive. This requirement on the weights can be achieved without loss of generality by calculating the optimal weights for classification and then applying this constraint (as a scale factor). The outputs from the two hidden layer neurons, denoted by $f_1(\underline{x})$ and $f_2(\underline{x})$, where $\underline{x}$ denotes the input P_1 neuron values, are

$$f_1(\underline{x}) = \|\tilde{w}_{11} x_1 + \tilde{w}_{12} x_2 + \tilde{w}_{13}\|^2 \tag{4}$$

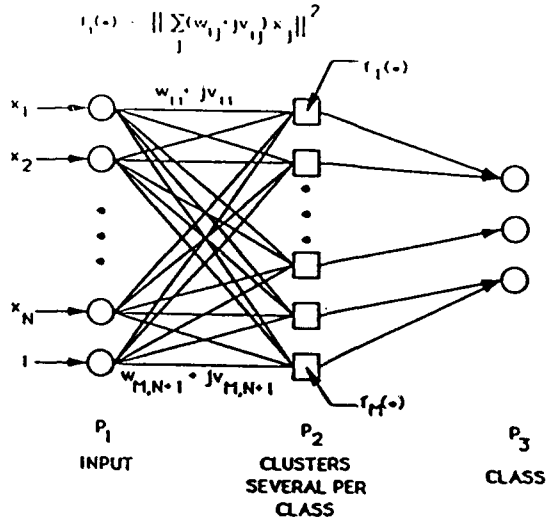


Figure 1: Piecewise Quadratic Neural Network

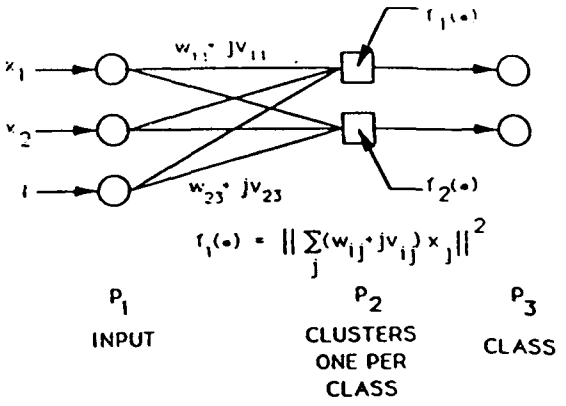


Figure 2: 2-class, 2-feature PQNN

$$f_2(\underline{x}) = \|\bar{w}_{21}x_1 + \bar{w}_{22}x_2 + \bar{w}_{23}\|^2 \tag{5}$$

Rewriting the complex-valued weight $\bar{w}_{ij}$ as in (3), the hidden layer neuron outputs become

$$\begin{aligned} f_1(\underline{x}) = {} & x_1^2(w_{11}^2 + v_{11}^2) + x_2^2(w_{12}^2 + v_{12}^2) + 2x_1x_2(w_{11}w_{12} + v_{11}v_{12}) + \\ & 2x_1(w_{11}w_{13} + v_{11}v_{13}) + 2x_2(w_{12}w_{13} + v_{12}v_{13}) + (w_{13}^2 + v_{13}^2) \end{aligned} \tag{6}$$

$$\begin{aligned} f_2(\underline{x}) = {} & x_1^2(w_{21}^2 + v_{21}^2) + x_2^2(w_{22}^2 + v_{22}^2) + 2x_1x_2(w_{21}w_{22} + v_{21}v_{22}) + \\ & 2x_1(w_{21}w_{23} + v_{21}v_{23}) + 2x_2(w_{22}w_{23} + v_{22}v_{23}) + (w_{23}^2 + v_{23}^2) \end{aligned} \tag{7}$$

To describe the discriminant surface (discriminant function) between classes 1 and 2, we equate $f_1(\underline{x})$ and $f_2(\underline{x})$ and obtain

$$\begin{aligned} & x_1^2(w_{11}^2 + v_{11}^2 - w_{21}^2 - v_{21}^2) + x_2^2(w_{12}^2 + v_{12}^2 - w_{22}^2 - v_{22}^2) + \\ & 2x_1x_2(w_{11}w_{12} + v_{11}v_{12} - w_{21}w_{22} - v_{21}v_{22}) + \\ & 2x_1(w_{11}w_{13} + v_{11}v_{13} - w_{21}w_{23} - v_{21}v_{23}) + \\ & 2x_2(w_{12}w_{13} + v_{12}v_{13} - w_{22}w_{23} - v_{22}v_{23}) + \\ & (w_{13}^2 + v_{13}^2 - w_{23}^2 - v_{23}^2) = 0 \end{aligned} \tag{8}$$

The general form for the decision surfaces in (8) is

$$ax_1^2 + bx_2^2 + cx_1x_2 + dx_1 + ex_2 + f = 0 \tag{9}$$

With different values for the a,b,c,d,e, and f coefficients, different surfaces can be produced. For example, if $c = d = e = 0$ and $a = b$, we obtain a circle with radius $\sqrt{f}$. If $c = d = e = 0$ and $a \neq b$, we obtain an ellipse; if $a = b = c = 0$, we obtain a straight line, given by

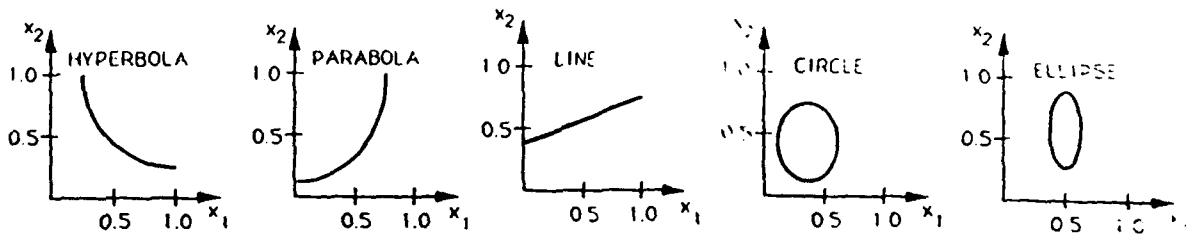


Figure 3: Possible decision surfaces for the two class, two-feature problem

$x_1 = -(e/d)x_2 + (f/d)$. Similarly, we can select the coefficients to produce a parabolic or hyperbolic surface.

As Eqs. 8 and 9 show, we can produce any general second order discriminant surface through various choices of the interconnection weights. When more than 2 features used, these become multidimensional surfaces and decision surfaces become piecewise hyperquadratic surfaces. If a given problem only requires a hyperplane, the weights automatically accomodate this (by choosing the appropriate real and imaginary parts of the weights to be zero). Therefore this architecture automatically accomodates any lower order surface. Fig. 3 shows the five basic decision surfaces possible with the PQNN. As the specific $\tilde{w}_{ij}$ values change, the parameters and locations of the various quadratic surfaces change.

4 Neural Network Algorithm

In the synthesis of the weights during training, we select prototypes of each class. We then select initial quadratic decision boundaries between sets of prototypes. These determine our initial complex-valued P_1-to-P_2 weights. We then use NN techniques to adapt these weights. We do this by determining the most active true and false class P_2 neurons and adapting their weights (after each presentation of all training samples) using gradient descent optimization. This is attractive as it uses pattern recognition techniques to select the initial weights (these are a much better choice and closer to the optimal weights than the typical choice of a random set of weights). The update algorithm then allows the NN to combine individual quadratic decision surfaces into the final piecewise quadratic decision surfaces used in classification.

We allow more than one prototype per class. The prototypes chosen implicitly contain the class distribution information (the class distributions are not explicitly calculated, but are used implicitly). The initial weights are not the prototype locations in feature space (as they were in ACNN).

5 Case Study

To demonstrate the performance of our PQNN architecture and algorithm and to allows its decision surfaces to be visualized, we consider a 2-class, 2-feature example (using the NN in Fig. 2). As our case study, we generated 500 samples of each of two classes. Fig. 4 shows 100 of the samples in each class. Discrimination of these two classes clearly requires nonlinear decision boundaries. We used only two hidden layer neurons (one per class) for this example, but more

than one per class can be used if necessary. The prototypes defined the initial P_1-to-P_2 weights. These were adapted in 50 iterations. Fig. 5 shows the resulting decision surface. It achieves $P_c = 97.16\%$ classification accuracy. In this case the decision surface is a second order surface (a circle). This is a true circle and not a piecewise approximation to it (as is produced with the ACNN and other NNs). The algorithm can produce any quadratic (second order) decision surface (e.g., circle, ellipse, parabola, hyperbola, etc.) as well as lower order (linear) decision surfaces (lines in two dimensional feature space and hyperplanes in higher dimensional feature spaces). The choice is automatic in the algorithm (as it selects the appropriate non-zero real and imaginary parts of each interconnection weight). For comparision, the ACNN was tested

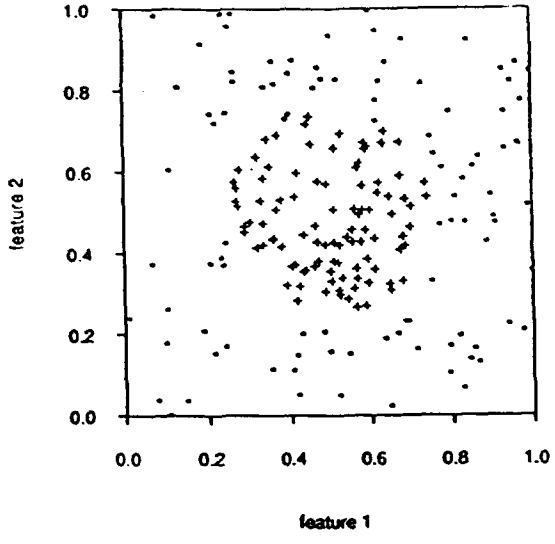


Figure 4: 2-class, 2-feature case study data ('+'=Class 1, '*'=Class 2)

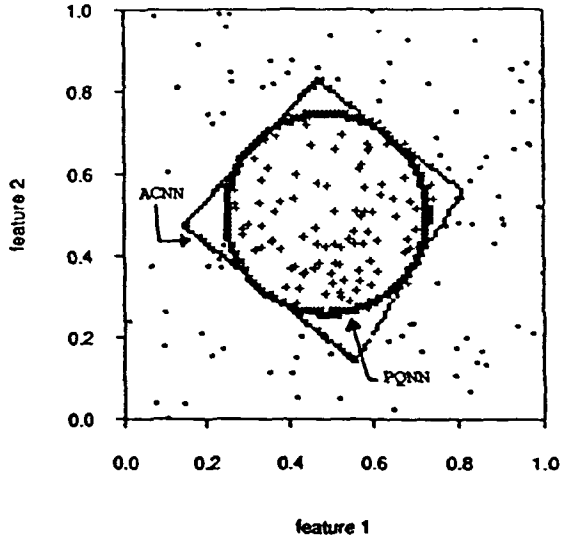


Figure 5: Decision boundaries formed by the PQNN and ACNN

with different numbers of P_2 neurons. With 12 neurons, piecewise linear decision surfaces were produced and this gave a classification accuracy 96.70%. The decision surface produced is also shown in Fig. 5. The number of P_2 neurons required in the ACNN is not easily determined except by extensive tests and the minimum number is quite critical. The PQNN does not have this disadvantage. Other NNs can produce similar decision surfaces at the expense of many more P_2 neurons and a larger interconnection mask. Also, selecting the number of P_2 neurons can be difficult. Clearly, the PQNN can produce exact quadratic decision surfaces (rather than piecewise ones) and can achieve this much more easily and automatically.

6 Discussion

A piecewise quadratic neural network (PQNN) was described and demonstrated. It achieves piecewise quadratic decision surfaces by the use of complex-valued interconnection weights and intensity detectors. An optical realization architecture was described and initial results presented.

The PQNN can be viewed as a higher order NN. However, it is very different from the

Giles, etal. [7] higher order NN, which used higher order neurons (x_1x_2, x_1^2, etc.) to achieve shift-invariance at the cost of N^4 interconnections (versus our use of complex-valued weights and square law detectors), and the Psaltis [8] higher order NN which used "higher order" to refer to 3-D optical volume holographic interconnections. The Neocognitron [9] uses many neuron layers to achieve shift-invariance and distortion-invariance (a form of higher order NN). We use a feature space neuron representation space to achieve shift and distortion-invariance (with much fewer neurons and interconnections). Our higher order NN is intended to produce more complicated decision surfaces (using only 2 or 3 neuron layers and simple matrix-vector operations, rather than volume holograms or multiple neuron layers).

7 Acknowledgements

This work was supported by a contract (DAAH01-89-C-0418) from the Defense Advanced Research Project Agency, monitored by the U.S. Army Missle Command.

References

[1] K. Hornick, *et al.*, "Multilayer feedforward networks are universal approximators," Neural Networks 2, 359-366, 1989.

[2] D. Casasent and E. Barnard, "Adaptive-clustering optical neural net," Applied Optics 29, 2603-2615, 1990.

[3] K. Wagner and D. Psaltis, "Multilayer optical learning networks," Applied Optics 26, 5061-5076, 1987.

[4] H. Yoshinaga, K-I. Kitayama, and T. Hara, "All-Optical Error-Signal Generation for Backpropagation Learning in Optical Multilayer Networks," Optics Letters 14, 202-204, 1989.

[5] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning internal representations by error propogation," in *Parallel Distributed Processing*, MIT Press, Cambridge, 1986.

[6] D. Casasent and F. Coetzee, "Error Diffusion Multilevel Phase Encoded CGH Elements," Proc. SPIE, Vol. 1347, July 1990.

[7] C.L. Giles, R.D. Griffen, and T. Maxwell, "Encoding Geometric Invariances in Higher-Order Neural Networks," Neural Information Processing Systems, D. Anderson, ed., AIP, Denver, CO, 301-309, 1988.

[8] D. Psaltis, C.H. Park, and J. Hong, "Higher Order Associative Memories and their Optical Implementations," Neural Networks 1, 149-163, 1988.

[9] K. Fukushima and S. Miyake, "Neocognitron: A new algorithm for pattern recognition tolerant of deformation and shifts in position," Pattern Recognition 15, 455-469, 1982.

IJCNN'92
Baltimore, MD
July 1992

Accuracy Effects in Pattern Recognition Neural Nets

David Casasent
Carnegie Mellon University
Center for Excellence in Optical Data Processing
Department of Electrical and Computer Engineering
Pittsburgh, PA 15213

Abstract

Various error sources (including analog accuracy, nonlinearities, and noise) are present in all neural nets. We consider their effects in training and testing on two different pattern recognition neural nets. We show that the neural nets considered allow some such effects to be included inherently in the neural net synthesis algorithm and that the effect of the other error sources can be "trained out" by proper selection of neural net design parameters. We consider multiclass distortion-invariant pattern recognition neural nets. Our results are applicable to analog VLSI and optical neural nets.

1. Introduction

For the difficult pattern recognition problems considered, the neural nets we used are reviewed in Section 2 together with the error sources considered and the algorithmic and training techniques considered to overcome these effects. Test results are included in Sections 3 and 4 and our conclusions are advanced in Section 5. We consider good probability of correct recognition (P_C'%) and large storage capacity (handling many classes M of objects with few neurons N_1) for distorted objects to be necessary performance goals.

2. Pattern Recognition Neural Nets

The three layer neural net (NN) architecture of Figure 1 is considered with N_1 input P_1 neurons, N_3 hidden layer P_3 neurons and $N_4 = C$ (the number of object classes) output P_4 neurons. The N_1 input neurons are a feature space representation (wedge sampled magnitude Fourier samples) to reduce dimensionality, training set size, and training time [1]. The number of neurons N_3 is selected by clustering techniques as prototype exemplars. These define initial P_1-to-P_3 weights which are then adapted in training. An important aspect of this NN is the use of a perceptron criterion (error) function with a parameter S (Figure 2). Selection of S achieves generalization, avoids overtraining, and is of use when noise and error effects are considered. Once the weights have been designed, this is a single-pass feed-forward NN in classification (recall) with no iterations used. This adaptive clustering NN (ACNN) has been detailed earlier [2].

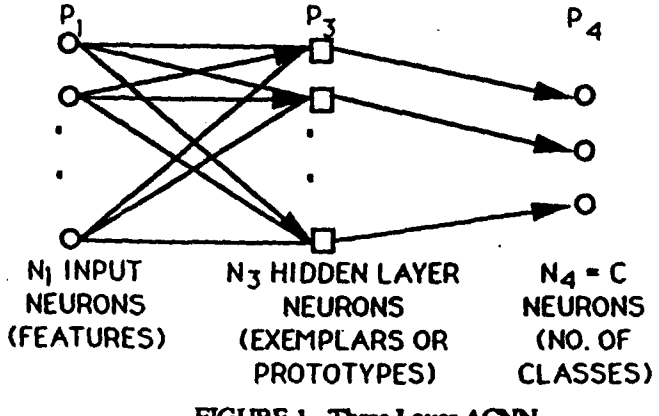


FIGURE 1. Three Layer ACNN

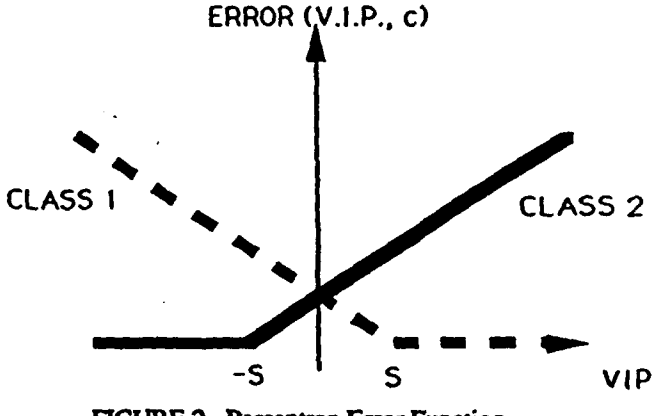


FIGURE 2. Perceptron Error Function

The P_1-to-P_3 operations are a matrix-vector multiplication implemented by a matrix $\underline{M}$ at plane P_2 operating on

an input P_1 neuron vector $\underline{x}$ to yield the P_3 vector $\underline{y}$ of neurons. We can address the storage capacity of the system using associative processor (AP) terminology where the P_1 inputs are keys, the P_3 vector (length K) is a recollection and the storage M is the number of key/recollection vector pairs stored. If each $\underline{x}$ input is assumed to be associated with a different $\underline{y}$ output at P_3, then $M = 2N_1$ is the largest storage possible and this can be achieved only with a Ho-Kashyap (HK) algorithm to compute the weights $\underline{M}$ as detailed elsewhere [3]. A robust HK algorithm [3] is of particular use in our present accuracy, noise and error source considerations. This uses $\underline{M}_I = \underline{Y}\,\underline{X}^T(\underline{X}\,\underline{X}^T + M\sigma^2\underline{I})^{-1}$ as the initial set of matrix weights in the synthesis algorithm where $\underline{X}$ and $\underline{Y}$ are matrices with all $\underline{x}$ and $\underline{y}$ vectors as their columns. When the parameter σ^2 is varied, this algorithm has been shown to provide best performance for input noise with variance σ^2. We have also related this σ value to the analog accuracy 2^B of a B-bit neural net as

$$\sigma = 2^{-B}/(12)^{1/2} \qquad (1)$$

as we will show. Thus, this technique has a sound theoretical basis [3].

The error sources we consider are listed in Table 1 with respect to the data plane P_1-to-P_3 they affect. Some entries require discussion. We consider a piecewise nonlinear error model for the input neurons (versus the ideal linear function) and we consider minimum (non-zero) off levels for P_1 and P_2 (due to light leakage in an optical system, etc.). The 6 bit input neuron accuracy assumed is typical of a 1% analog system. The P_2 accuracy in the neuron weights is B = 6 bits in standard analog VLSI (the same as for the P_1 input neurons) but can easily be higher (12 bits) in an optical system (since a fixed film mask with accurate encoding can be used). Advanced analog VLSI techniques can provide 10 bit P_1 and P_2 accuracies (at an increase in cost) while 6 bits is typical of standard fabrication methods. Error sources we found to be negligible are noted by an asterisk.

3. Ho-Kashyap Storage Capacity M/N Test Results

For our H-K algorithm tests, we use $N_1 = N = 16$ input neurons with random analog values and $N_3 = K = 8$ binary neurons at P_3. The maximum possible storage (M) for any NN is M = 2N or 32 for our case. Thus, we now consider the performance P_C% of H-K NNs that give M > N storage for various designs (σ values) with different error sources and bit accuracies present.

We first show in Table 2 that use of σ in our robust NN provides better P_C% results. In Test 1, the standard NN with the weights calculated with infinite accuracy data gave $P_C = 82.3\%$ correct classifications of input data when tested with a large $\sigma = 0.1$ amount of noise added to the input neuron values during tests. When the input data was quantized to 6 bits, noise tests gave similar 83% results. However, significantly better noise test results occurred ($P_C' \approx 91\%$) when the NN weights were synthesized with $\sigma = 0.1$ using our robust algorithm. Thus, our robust algorithm provides better results in noise and with limited accuracy. Much better P_C occurs (Tests 2-4) when the NN is tested with only limited accuracy inputs and weights (6-14 bits) without the large $\sigma = 0.1$ amount of input noise present and with $\sigma = 0.0045$ (for 6 bits), $\sigma = 0.0028$ (for 10 bits), etc. calculated from (1) used in synthesis of the NN. Thus, our robust NN algorithm gives excellent optimum results for limited input and weight accuracies with M > N (P_C% degrades as storage M/N increases as expected). We note (Tests 2-4) that no improvement in performance result as the number of bits of accuracy B was increased from 10 to 14 bits.

We then ran a number of NNs with different combinations of the error sources in Table 1 present and found that the major error sources were the input and weight accuracies and the nonlinear input neuron curve with the nonlinear input neuron curve being the major error source (as is expected with analog input neurons). We then analyzed various hidden layer P_3 encoding schemes to determine which gave the best P_C when the nonlinear input neuron errors were present. We found that L-max hidden layer neuron encoding [4] was best (in L-max encoding, the L = 2 most active P_3 neurons are found by a WTA and which two are activated defines the output P_4 class neuron activated). For M = 16 stored vector pairs (M/N = 1) we achieved excellent $P_C' = 98.8\%$ results and a lower $P_C' = 90.5\%$ with more storage (M = 20 or M/N = 1.375). We then quantified the input and weight accuracy required

for storage of M = 18 vector pairs. We found that B ≥ 4 bit input accuracy and B ≥ 8 or 10 bit weight accuracy was sufficient. When nonlinear P_1 neuron errors were present, 8 bit weights sufficed and without nonlinear P_1 errors no improvement occurred for ≥ 10 bit weights and P_C'% approached 100%. Thus, NNs require more weight accuracy than input accuracy and hence optical NNs have an advantage over analog VLSI implementations.

4. ADAPTIVE CLUSTERING NN TEST RESULTS

We then tested the ACNN design in Figure 1 for a multiclass distortion invariant pattern recognition problem involving 3 classes of aircraft with severe ±90° distortions in roll, pitch and yaw. The NN was trained on 1890 distorted inputs and tested on 1734 distorted inputs not present in the training set. We used N_1 = 16 input neurons, N_4 = C =3 output neurons (the number of object classes) and generally N_3 = 8 hidden layer neurons. We varied the NN parameter S (Figure 2) and the training procedure to achieve the best P_C'% when the major error sources were present (input and weight accuracies and nonlinear input neurons). In our data, we also note the minimum distance separation S′ between the cluster centers of the N_3 prototype hidden layer neurons and how this affects the parameter S used in NN synthesis.

We first calculated the weights using a full 32-bit accuracy digital processor and then tested the NN with full accuracy inputs and weights, with only the inputs quantized to different numbers of bits, and with only the weights quantized. Similar P_C% results occurred for both cases and for the training and test data. The S = 0.063 data in Table 3 shows averaged results that indicate that 4 bit input and weight accuracies give little degradation in P_C%. In these data, we used S > S′ = 0.030 since one might assume that a larger S forced separation would improve results. This is not the case since the S = S′ = 0.030 data in Table 3 shows better P_C'% results. We attribute this to the fact that a smaller S value allows decision boundaries to be closer to data clusters and that quantizing input data (after the weights have been calculated with high accuracy data) does not significantly move the data from the original clusters. Thus, we select S = S′ in our NN design.

In Table 3, the performance is excellent, but we still observe a loss in P_C'% as the input accuracy in testing is reduced (P_C'% decreases from 94.1% to 91.6% as P_1 decreases from 10 to 4 bits). To obtain improved results, we trained the neural net on quantized inputs. We also calculated new N_3 prototype neuron inputs and a new S = S′ value with these quantized inputs (in other words, full training was done on the low accuracy neural net to be used in testing). Table 4 shows our results using inputs and weights with only 4 bits of accuracy. The new S = S′ value was 0.070. As seen, the system with full accuracy gave P_C' ≈ 91.5%. When these calculated weights were quantized to 4 bits performance degraded by 2.8% to P_C' ≈ 88.7%. However, when training was performed on the low accuracy 4 bit system we obtained even better performance (P_C' ≈ 95.3%) than in the original system. Similar trends were obtained in all cases tested. Clearly, this NN allows NN accuracy effects to be trained out.

Similar results (Table 5) were obtained when input neuron nonlinearity (NL) errors were considered. Tests with this error source gave poor (P_C = 77.2%) results while training the NN with this error source present resulted in greatly improved (P_C' = 93.7%) test results. Thus, nonlinear error sources can also be trained out.

With a low accuracy NN, we expect more hidden layer neurons N_3 to improve P_C'% (more prototypes or data clusters are preferable since their separation decreases with accuracy) but if N_3 is made too large, local minima and increased training time result and S′ decreased. Table 6 shows that improved P_C'% results as N_3 is increased from 8 to 20 for a 32 bit accurate system (Tests 1 and 2) and for a 6-bit accurate system (Tests 3 and 4) with S = S′ used in all cases. Thus, increasing the number of hidden layer neurons improves the performance of low accuracy NNs.

	LOCATION	ERROR
	P_1	Nonlinear transfer function
*	P_1	Bias and gain spatial variation
	P_1	Accuracy (6 bits)
*	P_1	Off (zero) minimum level
*	P_2	Off (zero) minimum level
	P_2	Accuracy (6-12 bits)
*	P_2	Nonuniform beam collimation
*	P_2	Spatial (bias) and temporal (shot) noise

TABLE 1: Optical System Error Sources

TEST	B-BIT ACCURACY	P_C'% STD. NN	P_C'% ROBUST NN	REMARKS
1	∞ bits 6 bits	82.3 83.0	90.6 91.5	High $\sigma = 0.1$ noise
TEST	B-BIT ACCURACY	M/N	P_C'%	REMARKS
2	6 bits 10-14 bits	1 1	98.5 100	σ equal to bit accuracy
3	6 bits 10-14 bits	1.125 1.125	98.6 99.4	
4	6 bits 10-14 bits	1.375 1	98.0 99.5	

TABLE 2: Tests showing σ_{sys} improves P_C'% for σ = 6-bit value with high $\sigma = 0.1$ noise (Test 1) and with $\sigma = 2^{-B}/(12)^{1/2}$ (Tests 2-4)

	S = 0.030	S = 0.063
INPUT BITS	P_C'%	P_C'%
10	94.12	84.72
6	94.12	84.37
5	92.73	84.26
4	91.58	83.33
3	83.79	76.53
2	63.73	53.06
1	57.84	55.02

TABLE 3: Selecting $S = S' = 0.30$ yields better P_C with input P_1 quantization

	Full Accuracy Training (32 Bits)	Quantize Full Accuracy Weights to 4 Bits	Train on 4 Bit Inputs
P_C% (training)	91.3%	88.6%	95.1%
P_C% (testing	91.6%	88.8%	95.4%

TABLE 4: Improve P_C by training on a low accuracy NN

Scenario	P_C'% (Testing)
Test with NL input	77.2%
Train with NL input	93.7%

TABLE 5: Train-Out Input Neuron Nonlinearities to Improve P_C%

5. SUMMARY

We have showed that two advanced pattern recognition neural nets achieve excellent pattern recognition performance with various analog VLSI and optical neural net error sources. The HK neural net allows the best performance and storage in noise and with a limited accuracy NN. WTA hidden layer neurons are preferable when nonlinearity errors are present. We can training out limited neuron and weight accuracies and analog neuron nonlinearities. The advanced neural nets noted and proper use of their synthesis parameters (σ, S, number of hidden neurons equal to number of data clusters) allows these attractive error source properties.

ACKNOWLEDGMENTS

The author thanks L. Neiberg and S. Natarajan for obtaining, respectively, the HK and adaptive clustering neural net data presented.

REFERENCES

1. D. Casasent and E. Barnard, "Adaptive Clustering Neural Net for Piecewise Nonlinear Discriminant Surfaces", IJCNN'90 (*International Joint Conference on Neural Networks*), IEEE Catalog No. 90CH2879-5, June 1990, San Diego, Vol. I, pp. I-423 - I-428.
2. D. Casasent and E. Barnard, "Adaptive Clustering Optical Neural Net", Applied Optics, Vol. 29, pp. 2603-2615, 10 June 1990.
3. B. Telfer and D. Casasent, "Ho-Kashyap Optical Associative Processors", Applied Optics, Vol. 29, pp. 1191-1202, 10 March 1990.
4. D. Casasent and B. Telfer, "High Capacity Pattern Recognition Associative Processors", accepted for publication, Neural Networks.

CHAPTER 7: SIMULATION AND OPTICAL LABORATORY NEURAL NET RESULTS

7.1 ERROR-FREE SYSTEM TESTS

We conducted two error-free synthetic case studies and an aircraft case study comparing our new PQNN to other classifiers. Our first case study was a 2-class synthetic data set with 3 input neurons (features), with 500 samples in class 1 and 500 in class 2. We tested various classifiers with different numbers of hidden layer neurons N_3. Table 1 lists our P_C results. We see that all classifiers (except exemplars) perform well when N_3 is chosen large enough (and with the proper S value). The key result is that our PQNN performs approximately best for any N_3 choice and especially for smaller N_3 values.

		P_C (%)				
N_3	S	Exemplars	ACNN	Gaussian	BP	PQNN
2	0.25179	56.2	62.8	99.2	80.3	99.9
3	0.13190	66.5	62.4	95.1	80.7	96.9
4	0.04171	60.1	80.6	95.4	99.6	98.5
5	0.04171	68.9	80.8	97.8	99.5	98.8
6	0.01042	68.4	93.4	97.8	99.8	99.5
7	0.01042	66.3	93.3	97.8	99.5	99.1
8	0.01042	69.6	96.5	98.1	99.6	98.7
9	0.01042	77.3	98.8	97.9	99.7	98.6
10	0.00502	80.5	98.9	98.3	99.5	99.2

Table 1: P_C vs. N_3 (2 Class Case Study)

Figure 1 shows the original 2 class data. Figure 2 shows the different decision surfaces produced with the different classifiers with $N_3 = 2$. Clearly, the PQNN performs best (the Gaussian classifier performs similarly, since the data is Gaussian distributed).

Table 2 shows similar results for our second synthetic data set. This involved a 4-class problem (with 5000 total samples: 1020, 135, 1439 and 2406 samples in each class). We use $N_1 = 3$ input neurons (two features). The PQNN performs best for any N_3 and its advantage in P_C is larger for smaller N_3. Figure 3 shows the original data and Figure 4 shows the decision boundaries produced with $N_3 = 5$ hidden layer neurons.

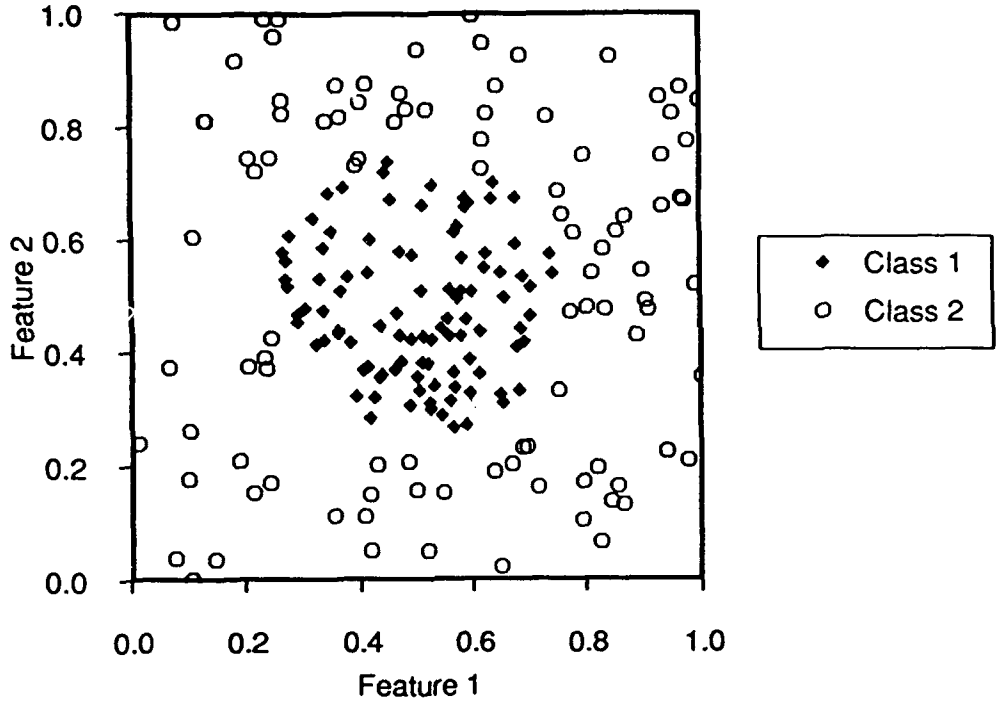
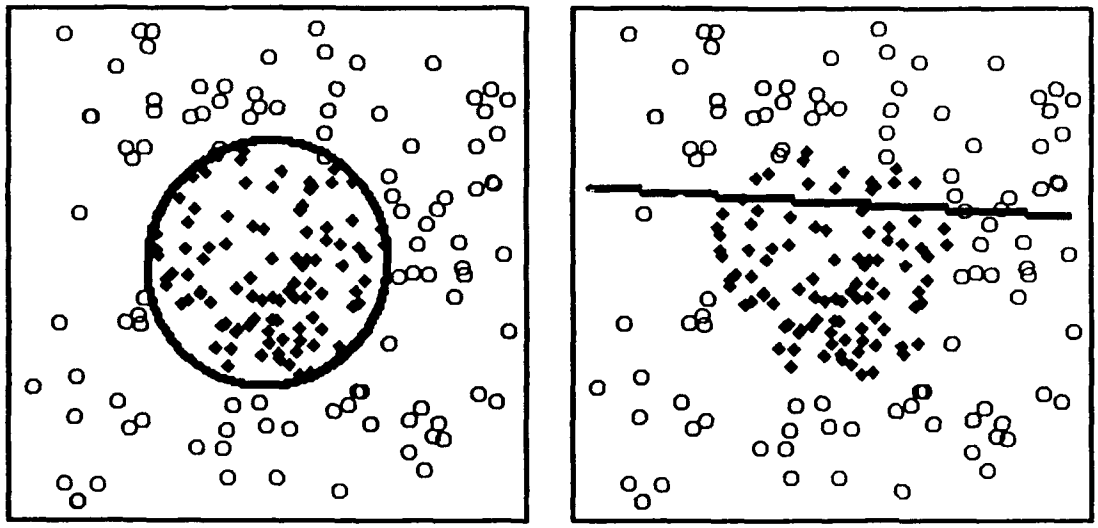


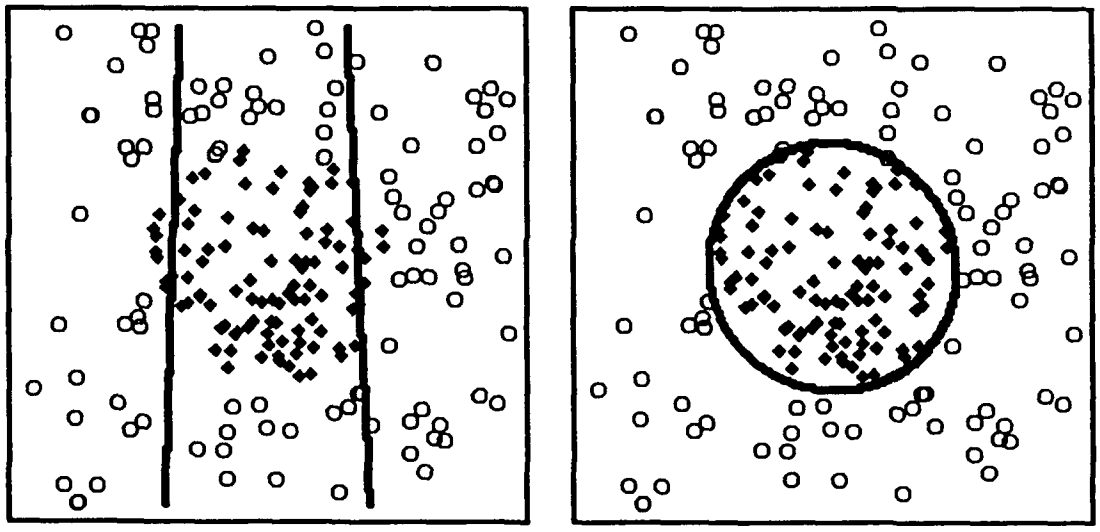
Figure 1: 2 Class Case Study Data

N_3	S	P_C (%)				
		Exemplars	ACNN	Gaussian	BP	PQNN
4	0.04503	43.1	74.9	85.9	92.4	96.8
5	0.04503	51.5	74.7	85.7	94.6	95.2
6	0.03154	62.2	75.0	83.5	94.8	96.4
7	0.02380	67.8	75.7	91.0	94.5	95.2
8	0.02380	69.4	94.2	91.3	96.0	96.1
9	0.00935	69.9	94.3	92.0	94.8	96.7
10	0.00935	69.8	94.4	90.6	95.2	96.8
11	0.00935	72.5	94.4	88.4	95.6	94.9
12	0.00032	71.9	94.8	91.5	95.8	95.3
13	0.00032	72.3	93.9	93.7	95.4	96.9
14	0.00032	73.8	95.0	93.6	95.4	97.0
15	0.00032	74.3	93.0	94.0	95.0	96.6

Table 2: P_C vs. N_3 (4 Class Case Study)



(a) PQNN (b) ACNN



(c) Backprop (d) Gaussian

Figure 2: Decision Boundaries for 2 Class Case Study for $N_3 = 2$ hidden layer neurons.

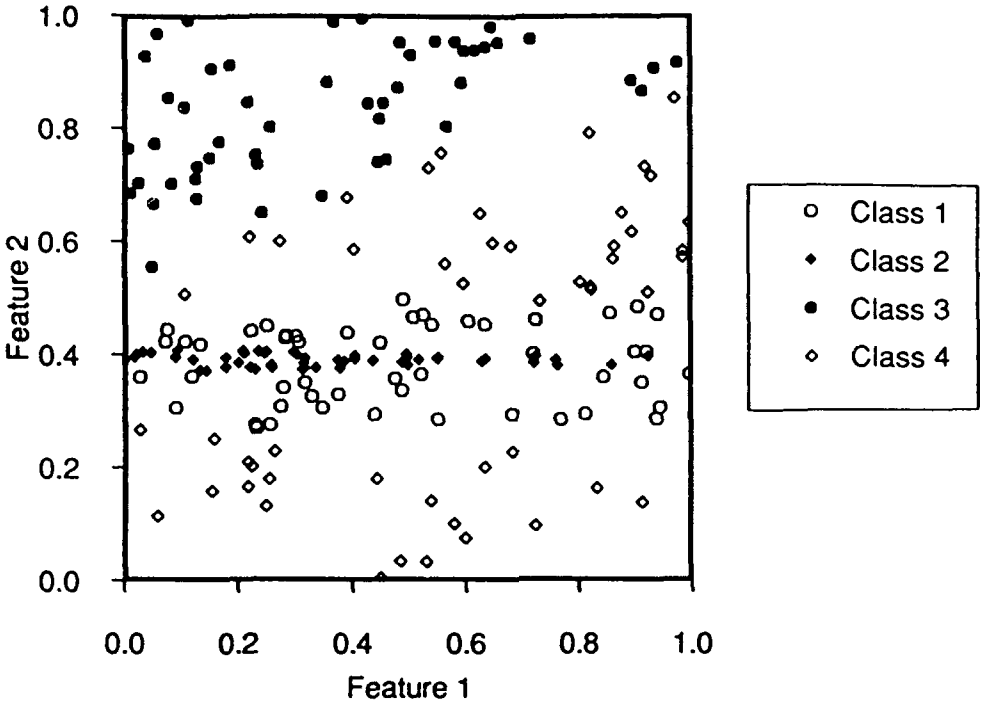


Figure 3: 4 Class Case Study Data

We also considered an aircraft database of 128×128 binary synthetic images of three aircraft (F-4, F-104, and DC-10). 630 images of each aircraft, representing different azimuth and elevation views, were used as training images. The azimuth angles range from -85° to +85° in 5° increments, while the elevation angles span 0° to 90° in 5° increments. There are 578 test set images per aircraft type at 2.5° intermediate angles. The features used are invariant to image translation. As inputs to the classifiers, we use 15 wedge-sampled Fourier transform features. The wedge samples are normalized so that they provide scale and shift invariance. The P_C performance versus N_3 showed that the PQNN is again best.

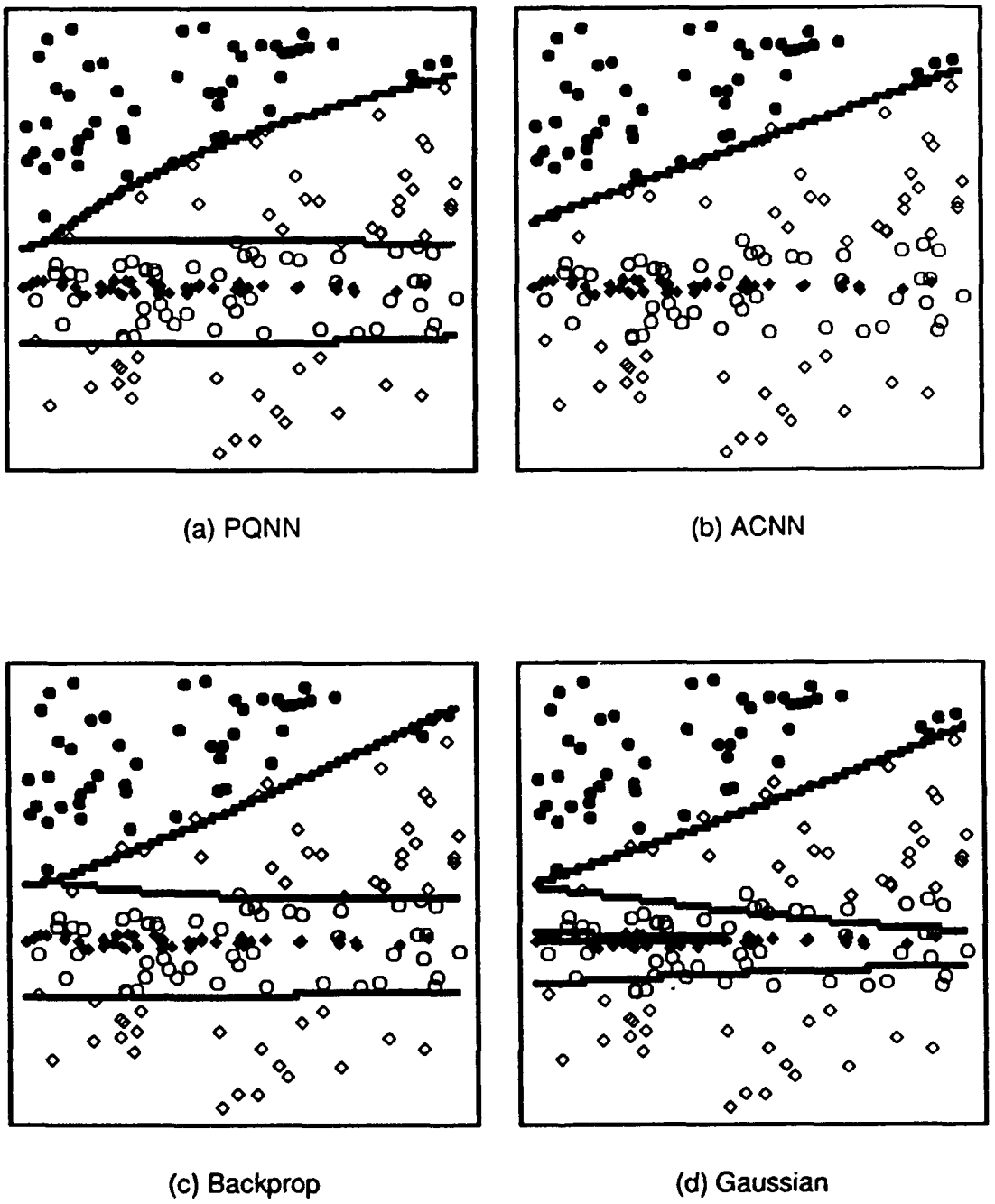


Figure 4: Decision Boundaries for 4 Class Case Study for $N_3 = 5$ hidden layer neurons.

7.2 OPTICAL LABORATORY SYSTEM NEURAL NET TESTS

We performed optical laboratory tests on our two-class synthetic and three-class aircraft case studies and compared results to those obtained by simulations. In all cases, we consider only real weights. This completes Tasks 7 and 8 concerning the ACNN and PQNN. Table 1 lists our results. For these 2 case studies, we show the optical laboratory data (Test 1) and that they agree with the simulation results when all errors were present and trained out (assuming 5-bit LCTV input neuron accuracy). Thus, the validity of our simulator and our error source models are verified, as is our training out algorithm. If 6-bit input neuron accuracy were available, better P_C would result as shown in Test 3 versus Test 4 (57.2% versus 56.8% and 90.5% versus 88.7%). For comparison, we list the P_C obtained with an ideal system (Test 2) and see that it is not significantly better than the results we obtained. Thus, our present neural net is nearly the best possible and is quite useful. Our training out algorithm clearly provides much better P_C results.

Test		2 Class Aircraft Case Study	3 Class Aircraft Case Study	
			Test	Train
1	Optical lab	56.5	88.3	86.1
2	Ideal (no errors)	62.8	84.7	83.6
3	All errors present Trained out (6-bit input neurons)	57.2	90.5	91.1
4	All errors present. Trained out (5-bit input neurons)	56.8	88.7	86.4

Table 3: P_C results for the PQNN with real weights for several case studies using the optical laboratory system, an ideal system and systems with different levels of input accuracy using our training output algorithm.

7.3 OPTICAL LABORATORY HIGH CAPACITY HK NN TESTS

We used our 1:1 Ho-Kashyap neural net to demonstrate its high storage and its superior performance. The system parameters are given in Table 4. The error sources considered are given in Table 5. We trained out all errors except errors 8 and 9. We corrected for errors 5 and 6. No input noise was present. The σ_{syn} control parameter used ($\sigma_{syn} = 0.00025$) corresponded to 10-bit accuracy.

Simulations predicted a recall accuracy of $P'_C = 98.17\%$ (average accuracy of 100 random sets of input data) for a neural net with the parameters as listed in Tables 4 and 5. For the exact neural

net run in the optical lab with the specific M = 24 input vectors used, simulations (with only one run) gave $P'_C = 100\%$. We ran the system in the laboratory and achieved $P'_C = 23/24 \approx 96\%$. This corresponds to only one error in the 24 output vectors and the single output vector that was incorrect had only one of its output elements wrong. We attribute this error to the space-variant beam profile which is only approximately modeled by a centered Gaussian. This is excellent storage performance density (more than any other neural net) and our simulations and optical lab results match well and the use of our training out algorithm is verified.

PARAMETER	VALUE	DESCRIPTION
N	16	Input vector dimension (unipolar elements)
K	8	Output vector elements (bipolar elements)
M	24 (M/N = 1.5)	Vector pairs stored
L	3	L-max output vector encoding parameter
σ_1	0.0	No input noise was added to key vectors

Table 4: AP system parameters

Error	Location	Error Description	Parameter Value	Significant Error?
1	P_1	Nonlinear input device characteristics		YES
2	P_1	Input accuracy	5 bits	YES
3	P_1	Off (zero) minimum level	0.0006	NO
4	P_2	Mask accuracy	8 bits	YES
5	P_2	Mask nonlinear device characteristics		YES
6	P_2	Gaussian beam taper	15%	YES
7	P_2	On (1.0) maximum level	0.931	YES
8	P_3	Detector precision	10 bits	YES
9	P_3	Detector temporal shot noise variance	10^{-7}	NO

Table 5: Optical system error sources summary