

Iterative Improvement of a Nearest Neighbor Classifier

Hung-Chun Yau and Michael T. Manry

Department of Electrical Engineering
University of Texas at Arlington
Arlington, Texas 76019

Abstract

In practical pattern recognition applications, the nearest neighbor classifier (NNC) is often applied because it does not require an a priori knowledge of the joint probability density of the input feature vectors. As the number of example vectors is increased, the error probability of the NNC approaches that of the Bayesian classifier. However, at the same time, the computational complexity of the NNC increases. Also, for a small number of example vectors, the NNC is not optimal with respect to the training data. In this paper, we attack these problems by mapping the NNC to a sigma-pi neural network, to which it is partially isomorphic. A modified form of back-propagation (BP) learning is then developed and used to improve classifier performance. As examples, we apply our approach to the problems of hand-printed numeral recognition and geometrical shape recognition. Significant improvements in classification error percentages are observed for both the training data and testing data.

Send reprint requests to Prof. Michael T. Manry, Department of Electrical Engineering, University of Texas at Arlington, Arlington, Texas 76019. Phone: 817-273-3483.

Key Words: nearest neighbor classifier, sigma-pi network, back-propagation, character recognition, isomorphic classifiers, shape recognition, deformation-invariant features.

Nomenclature

D_{ij}	metric distance between vectors \mathbf{x} and \mathbf{r}_{ij} .
E_p	the error function for the p th input pattern.
E_T	the error function being minimized by back-propagation.
$G(i)$	the gradient of E_p with respect to $P_{net}(i)$.
$g(i,j)$	the gradient of E_p with respect to $S_{net}(i,j)$.
N_c	the number of classes.
N_f	the number of input dummy units in the neural net, and the number of elements in the feature vector \mathbf{x} .
N_p	the total number of training patterns.
N_s	the number of clusters per class.
$P_{in}(i)$	products of sigma-pi unit outputs of class i .
$P_{net}(i)$	net input of the i th output unit.
$P_{out}(i)$	output of the i th output unit.
$P_w(i,\hat{i})$	connection weights for $P_{in}(\hat{i})$ and $P_{net}(i)$.
$P\theta(i)$	threshold of the i th output unit.
\mathbf{r}_{ij}	the j th reference vector of the i th class.
$r_{ij}(k)$	the k th element of \mathbf{r}_{ij} .
$S_{net}(i,j)$	net input of the j th sigma-pi unit of the i th class.
$S_{out}(i,j)$	output of the j th sigma-pi unit of the i th class.
$Sw1(i,j,k)$	the connection weights for the first order of the k th feature to the j th sigma-pi unit of the i th class.
$Sw2(i,j,k)$	the connection weights for the second order of the k th feature to the j th sigma-pi unit of the i th class.
$S\theta(i,j)$	threshold of the j th sigma-pi unit of the i th class.
$T_{out}(i)$	the desired activation for the i th output unit.
\mathbf{x}	feature vector.
$x(k)$	the k th element or feature of \mathbf{x} .
\mathbf{v}_{ij}	the N_f by 1 variance vector for the j th cluster of the i th class.
$v_{ij}(k)$	the k th element of \mathbf{v}_{ij} .
ζ	learning factor in back-propagation.

I. Introduction

As pointed out by Duda and Hart (1973) and Fukunaga (1972), the NNC approximates the minimum error Bayesian classifier in the limit as the number of reference vectors gets large. When the feature vector joint probability density is unknown, the NNC would be the preferred classifier, except for two problems. First, a prohibitive amount of computation is required for its use. Second, the NNC's performance is usually not optimized with respect to the training data.

As more hardware for parallel processing becomes available (neural or conventional), the first problem will be solved. Neural networks have already been used to attack the second problem. Lippman (1987) has pointed out that the multi-layer perceptron is very similar to the NNC, and can be used as a classifier. As illustrated in Fig. 1, Kohonen (1990, 1988) and Kohonen et.al. (1988) have mapped the NNC to a neural network, which they have named learning vector quantization (LVQ). They have also suggested a learning rule for training the network. Hecht-Nielsen (1987, 1988) developed the two-layer counter-propagation network (CPN) that combines the Kohonen self-organization and Grossberg outstar algorithms. The LVQ and CPN networks are then isomorphic to types of NNC.

In this paper we develop techniques for optimizing the NNC through the use of a sigma-pi back-propagation network. In section II, we give a simple method for mapping the NNC's components to a sigma-pi neural network. The learning rule for the network is described in section III. We apply our algorithm to the improvement of classifiers for hand-printed numerals and geometric shapes in section IV. The weights, initialized by a direct mapping procedure instead of via random assignment, effectively shorten the learning procedure and increase the possibility that a global minimum of the error function can be reached. Conclusions are given in section V.

II. Nearest Neighbor Isomorphic Networks

In this section, we describe the nearest neighbor isomorphic network (NNIN), which is a BP network isomorphic to a type of NNC. This network uses fairly conventional sigma-pi units, described by Rumelhart et. al. (1986), in the hidden layer and units similar to the product units of Durbin and Rumelhart (1989) in the output layer. A set of good initial weights and thresholds can be directly determined from the reference feature vectors via appropriate mapping equations.

A. First Layer

Each pattern is represented by a feature vector, $\mathbf{x} = [x(1) \ x(2) \ \cdots \ x(N_f)]^T$, where N_f is the number of elements in the feature vector \mathbf{x} . Cluster mean vectors have been used for keeping a small number of most representative samples. For example, $\mathbf{r}_{ij} = [r_{ij}(1), r_{ij}(2), \cdots, r_{ij}(N_f)]^T$ is the mean vector of the j th cluster of the i th class. Let D_{ij} represent the distance between vectors \mathbf{x} and \mathbf{r}_{ij} . The first processing step in the NNC is the calculation of the D_{ij} numbers. For the NNC cluster discriminant, we use the Mahalanobis distance for independent features rather than

the Euclidean distance. Let \mathbf{v}_{ij} denote the feature variance vector for the j th cluster of the i th class and let $v_{ij}(k)$ denote its k th element. Then

$$D_{ij}^2 = \sum_{k=1}^{N_f} \frac{[x(k) - r_{ij}(k)]^2}{v_{ij}(k)} \quad (1)$$

As shown in Fig. 2, the first processing layer consists of $N_c \cdot N_s$ sigma-pi units which are connected to the N_f input features. Here N_c is the number of classes and N_s is the number of clusters per class. Let $S_{net}(i,j)$, $S\theta(i,j)$, and $S_{out}(i,j)$ denote the net input, threshold, and output of the j th unit of the i th class, respectively. $Sw1(i,j,k)$ and $Sw2(i,j,k)$ denote the connection weights from the k th input feature to the j th sigma-pi unit of the i th class. The sigma-pi network and activation are respectively

$$S_{net}(i,j) = S\theta(i,j) + \sum_{k=1}^{N_f} [Sw1(i,j,k)x(k) + Sw2(i,j,k)x^2(k)] \quad (2)$$

$$S_{out}(i,j) = \frac{1}{1 + \exp[-S_{net}(i,j)]} \quad (3)$$

Comparing $S_{net}(i,j)$ with D_{ij}^2 , we may assign the initial weights and threshold of the j th sigma-pi unit of the i th class as follows:

$$S\theta(i,j) = \sum_{k=1}^{N_f} \frac{r_{ij}^2(k)}{v_{ij}(k)} \quad (4)$$

$$Sw1(i,j,k) = -2 \frac{r_{ij}(k)}{v_{ij}(k)} \quad (5)$$

$$Sw2(i,j,k) = \frac{1}{v_{ij}(k)} \quad (6)$$

This first sigma-pi layer can also be mapped back to the NNC's first layer. Hence the first layer of the NNC and the first layer of sigma-pi network are isomorphic.

B. Second Layer

The second step in a NNC is the Min operation, which consists of finding i and j such that D_{ij} is minimized. The estimated class is then i . As seen in Fig. 2, the Min operation of the NNC has been replaced in the second processing layer by sigma-pi units of a new type. These are the product units of Durbin and Rumelhart (1989), with each feature raised to the first power. For a discussion of the merits of product units versus Min units, see Appendix A. Let $P\theta(i)$, $Pnet(i)$ and $Pout(i)$ denote the threshold, net input and output of the i th unit in the second layer. The $Pw(i,\hat{i})$ are the connection weights for $Pin(\hat{i})$ and $Pnet(i)$.

$$Pin(i) = \prod_{j=1}^{Ns} Sout(i,j) \quad (7)$$

$$Pnet(i) = P\theta(i) + \sum_{\hat{i}=1}^{Nc} Pw(i,\hat{i}) Pin(\hat{i}) \quad (8)$$

$$Pout(i) = \frac{1}{1 + \exp[-Pnet(i)]} \quad (9)$$

It is simple to initialize the second layer as $Pw(i,\hat{i}) = \delta(i-\hat{i})$ and $P\theta(i) = 0$. This is equivalent to replacing the Min operation by a product at the beginning of training. The output layer has desired activation values of 1 for incorrect classes and 0 for the correct class.

III. Learning Rule

In the previous section, we initialized our network with good initial weights. However, the net's performance can still be improved via a form of back-propagation, as described in Rumelhart, et. al. (1986). Let $Tout(i)$ be the desired output. Np denotes the total number of training patterns. Using the q -norm, which is the p -norm of Cheney (1982) and Powell (1981), the system performance measure is defined as

$$E_T = \sum_{p=1}^{Np} E_p \quad (10)$$

$$E_p = \left(\sum_{i=1}^{Nc} [Tout(i) - Pout(i)]^{2q} \right)^{1/q} \quad (11)$$

where E_p is the error measure for the p th input pattern, and q is a positive integer. In practice

we alter the error criterion from the least square approach ($q=1$) to the minimax approach ($q=\infty$) or vice versa when the learning process slows. In our experiments, this adaptive- q technique results in an increase in learning speed.

A. Gradients for $q=1$

For the $q=1$ case, the traditional back-propagation learning rule is applied to the isomorphic network. The gradient of E_p with respect to the net input of the second layer units is

$$G(i) = \frac{-\partial E_p}{\partial P_{net}(i)} = \frac{-\partial E_p}{\partial P_{out}(i)} \frac{\partial P_{out}(i)}{\partial P_{net}(i)} \quad (12)$$

$$= 2 [T_{out}(i) - P_{out}(i)] \cdot P_{out}(i) \cdot [1 - P_{out}(i)]$$

Then

$$\frac{-\partial E_p}{\partial P\theta(i)} = \frac{-\partial E_p}{\partial P_{net}(i)} \frac{\partial P_{net}(i)}{\partial P\theta(i)} = G(i) \quad (13)$$

$$\frac{-\partial E_p}{\partial Pw(i,\hat{i})} = \frac{-\partial E_p}{\partial P_{net}(i)} \frac{\partial P_{net}(i)}{\partial Pw(i,\hat{i})} = G(i) \cdot Pin(\hat{i}) \quad (14)$$

The gradients for the first layer units are

$$\begin{aligned} g(i,j) &= \frac{-\partial E_p}{\partial S_{net}(i,j)} \\ &= \sum_{\hat{i}=1}^{N_c} \frac{-\partial E_p}{\partial P_{net}(\hat{i})} \frac{\partial P_{net}(\hat{i})}{\partial S_{out}(i,j)} \frac{\partial S_{out}(i,j)}{\partial S_{net}(i,j)} \\ &= Pin(i) \cdot [1 - S_{out}(i,j)] \sum_{\hat{i}=1}^{N_c} G(\hat{i}) Pw(\hat{i},i) \end{aligned} \quad (15)$$

Then

$$\frac{-\partial E_p}{\partial S\theta(i,j)} = \frac{-\partial E_p}{\partial S_{net}(i,j)} \frac{\partial S_{net}(i,j)}{\partial S\theta(i,j)} = g(i,j) \quad (16)$$

$$\frac{-\partial E_p}{\partial Sw1(i,j,k)} = \frac{-\partial E_p}{\partial Snet(i,j)} \frac{\partial Snet(i,j)}{\partial Sw1(i,j,k)} = g(i,j) \cdot x(k) \quad (17)$$

$$\frac{-\partial E_p}{\partial Sw2(i,j,k)} = \frac{-\partial E_p}{\partial Snet(i,j)} \frac{\partial Snet(i,j)}{\partial Sw2(i,j,k)} = g(i,j) \cdot x^2(k) \quad (18)$$

For $1 \leq i, \hat{i} \leq Nc$, $1 \leq j \leq Ns$, $1 \leq k \leq Nf$, the thresholds and connection weights are updated as

$$P\theta(i) = P\theta(i) + \zeta \cdot G(i) \quad (19)$$

$$Pw(i, \hat{i}) = Pw(i, \hat{i}) + \zeta \cdot G(i) \cdot Pin(\hat{i}) \quad (20)$$

$$S\theta(i,j) = S\theta(i,j) + \zeta \cdot g(i,j) \quad (21)$$

$$Sw1(i,j,k) = Sw1(i,j,k) + \zeta \cdot g(i,j) \cdot x(k) \quad (22)$$

$$Sw2(i,j,k) = Sw2(i,j,k) + \zeta \cdot g(i,j) \cdot x^2(k) \quad (23)$$

where ζ is a learning factor.

B. Gradients for Infinite q

As q approaches infinity, the error measure for the pth input pattern can be written as

$$\begin{aligned} E_p &= \lim_{q \rightarrow \infty} \left(\sum_{i=1}^{Nc} [Tout(i) - Pout(i)]^{2q} \right)^{1/q} \\ &= \max_{1 \leq i \leq Nc} [Tout(i) - Pout(i)]^2 \\ &= [Tout(i_x) - Pout(i_x)]^2 \end{aligned} \quad (24)$$

where class i_x has the maximum error. In the training we modify the weights and threshold of the unit which has the maximum error for each pattern. For the pth pattern, the gradient for the output unit with maximum error is

The gradients for the first layer units are

$$G(i_x) = \frac{-\partial E_p}{\partial Pnet(i_x)} = \frac{-\partial E_p}{\partial Pout(i_x)} \frac{\partial Pout(i_x)}{\partial Pnet(i_x)} \quad (25)$$

$$= 2 [Tout(i_x) - Pout(i_x)] \cdot Pout(i_x) \cdot [1 - Pout(i_x)]$$

$$g(i,j) = \frac{-\partial E_p}{\partial Snet(i,j)} \quad (26)$$

$$= \frac{-\partial E_p}{\partial Pnet(i_x)} \frac{\partial Pnet(i_x)}{\partial Sout(i,j)} \frac{\partial Sout(i,j)}{\partial Snet(i,j)}$$

$$= Pin(i) \cdot [1 - Sout(i,j)] \cdot G(i_x) \cdot Pw(i_x,i)$$

Given i_x , the thresholds and connection weights are updated as

$$P\theta(i_x) = P\theta(i_x) + \zeta \cdot G(i_x) \quad (27)$$

$$Pw(i_x,i) = Pw(i_x,i) + \zeta \cdot G(i_x) \cdot Pin(i) \quad (28)$$

$$S\theta(i,j) = S\theta(i,j) + \zeta \cdot g(i,j) \quad (29)$$

$$Sw1(i,j,k) = Sw1(i,j,k) + \zeta \cdot g(i,j) \cdot x(k) \quad (30)$$

$$Sw2(i,j,k) = Sw2(i,j,k) + \zeta \cdot g(i,j) \cdot x^2(k) \quad (31)$$

for $1 \leq i \leq Nc$, $1 \leq j \leq Ns$, and $1 \leq k \leq Nf$.

C. Training

In order to speed up the training, we make the learning factor adaptive after each training iteration as

$$\zeta = \begin{cases} \alpha \cdot \zeta, & \text{if } \Delta E_T \geq 0 \\ \beta \cdot \zeta, & \text{if } \Delta E_T < 0 \end{cases} \quad (32)$$

where α is an assigned constant less than 1 and $\beta = (1/\alpha)^c$ for a positive integer constant c . We define $\Delta E_T(n) = \hat{E}_T(n) - \hat{E}_T(n-1)$ where n is the iteration number and the smoothed error is $\hat{E}_T(n) = (1/2)[E_T(n) + \hat{E}_T(n-1)]$ and $\hat{E}_T(0) = E_T(1)$. For example, we use $\alpha = 0.9$, $c = 5$ in our back-propagation training. If the smoothed error $\hat{E}_T(n)$ increases in a given iteration, the learning factor is decreased, as seen in (32). If $\hat{E}_T(n)$ decreases, the learning factor is increased. Note that the rate of decrease is much greater than the rate of increase, preventing the error from blowing up.

IV. Classification Experiments

In this section, we apply our forward mapping algorithms to two pattern recognition problems. The first problem is that of recognizing hand-printed numerals. The second is a geometrical shape recognition problem. To evaluate our network's effectiveness, we compare the recognition performances of our algorithm with those of two other well-known algorithms.

A. Hand-printed Numeral Recognition Example

Our raw data consists of images of ten hand-printed numerals, collected from 3,000 people by the I.R.S. (see Weideman et. al. 1989). We randomly chose 300 characters from each class to generate a 3,000 character training data set. A separate testing data set consists of 6,000 characters, 600 from each class, which are selected from the remaining database. Images are 32 by 24 binary matrices. An image scaling algorithm is used to remove size variation in the characters.

The feature set contains 16 elements, developed by Gong and Manry (1989), which are non-Gaussian. We use the iterative K-mean clustering algorithm, described by Anderberg (1973) and Duda and Hart (1973), to select 10 reference cluster mean and variance vectors for each class. This algorithm differs from the standard K-mean clustering algorithm in that each cluster is constrained to have the same number of members.

We compared the performance of the NNIN with that of the NNC, and LVQ2.1 for the ten numeral classes. We plot the recognition error percentage versus the training iteration number for the training data set for those different classifiers in Fig.3. Table 1 shows the final results for the training and testing data sets. The NNC is designed for ten numeral classes and it has 10.43 % classification error for the training data and 11.60 % classification error for the testing data. The NNC was mapped to the LVQ2.1 and NNIN networks, in order to initialize their weights. The LVQ2.1 network quickly reduced the error percentage and finally leveled off at 6.37 % error for the training set. The LVQ2.1 network decreased the error rate for the testing set down to 9.73 %. Initially, the error for the NNIN was 9.47 %. Although the error percentage was initially small, back-propagation with the adaptive-q technique improved the performance

Table 1 Recognition experiments for hand-printed numerals
(in error percentage)

Classifier	Training Set	Testing Set
NNC	10.43 %	11.60 %
LVQ2.1	6.37 %	9.73 %
NNIN	4.93 %	7.82 %

still further. After 150 training iterations, the NNIN classifier had 4.93 % classification error for the training data and 5.5 % error for the testing data. Both the LVQ2.1 and the NNIN performed much better than the NNC. However, the NNIN had better performance than LVQ2.1.

B. Shape Recognition Example

In order to confirm our results, we obtained a second database for a completely different problem; the classification of geometric shapes. The four primary geometric shapes used in this experiment are: (1) ellipse, (2) triangle, (3) quadrilateral, and (4) pentagon. Each shape image consists of a matrix of size 64×64 . Each element in the matrix represents a binary-valued pixel in the image. For each shape, 200 training patterns and 200 testing patterns were generated using different degrees of deformation. The deformations included rotation, scaling, translation, and oblique distortions. The weighted log-polar transform, which computes energies in several ring and wedge-shape areas in the Fourier transform domain, was applied to extract 16 features for each pattern. A detailed description of these features is given in Appendix B. In Fig.4 we plot the recognition error percentage versus the training iteration number for the training data set for the NNC, LVQ2.1, and NNIN respectively. The classifier performances are presented in Table 2 for the testing data set.

Table 2 Recognition experiments for geometric shapes
(in error percentage)

Classifier	Training Set	Testing Set
NNC	6.38 %	5.88 %
LVQ2.1	2.88 %	4.88 %
NNIN	0.00 %	1.63 %

Note that mapping the NNC to the NNIN increased the error rate in Fig. 4, contrary to

what is seen in the hand-printed numeral recognition experiment of Fig. 3. Nonetheless, the NNIN network reduces the misclassification rate very quickly and finally reaches 0 % classification error for the training data. As in the first experiment, the NNIN has better performance than the NNC and LVQ2.1 classifiers for both training and testing data sets.

V. Conclusions

In this paper we have shown that under some circumstances, a NNC can be mapped to a sigma-pi neural network and then improved through back-propagation training. This method was applied to NNC's for hand-printed numerals and geometric shapes. Although the error percentages were initially small, back-propagation in the corresponding neural nets improved the performances still further.

The two examples, hand-printed numerals and geometric shapes, show the range of performances possible with the NNIN. Mapping of the NNC to the NNIN greatly reduces the classification errors in both problems. Note that the use of the adaptive q-norm error criterion may cause the error percentage to abruptly rise or drop during training. However, we observe that the curves of error percentage versus iteration number still decrease asymptotically. Also this learning technique may help us avoid getting trapped in local minima.

Because the initial weights used are good, the error percentage quickly levels off. Since the classification performance of the network is improved for both training and testing data, it is apparent that the classifier is generalizing, and not just memorizing the training data.

VI. References

- Anderberg, M. (1973). Cluster analysis for applications. New York: Academic Press.
- Casasent, D. and Psaltis, D. (1976). Position, rotation, and scale invariant optical correlation. *Applied Optics*, Vol.15, pp.1795-1799.
- Cheney, E.W. (1982). Introduction to approximation theory. New York: Chelsea Publishing.
- Duda, R.O. and Hart, P.E. (1973). Pattern classification and scene analysis. New York: Wiley.
- Durbin, R. and Rumelhart, D.E. (1989). Product units: a computationally powerful and biologically plausible extension to backpropagation networks. *Neural Computation*, Vol.1, pp.133-142.
- Fukunaga, K. (1972). Introduction to statistical pattern recognition. New York: Academic ~~Pr~~
- Gong, W. and Manry, M.T. (1989). Analysis of non-Gaussian data using a neural network. *International Joint Conference on Neural Networks*, Washington D.C., June 1989, Vol.2, ~~176~~

- Hecht-Nielsen, R. (1987). Counterpropagation network. *IEEE First International Conference on Neural Networks*, San Diego, Ca., June 1987, Vol.2, pp.19-32.
- Hecht-Nielsen, R. (1988). Applications of counterpropagation network. *Neural Networks*, Vol.1, pp.131-139.
- Kohonen, T. (1990). Improved versions of learning vector quantization. *International Joint Conference on Neural Networks*, San Diego, Ca., June 1990, Vol.1, pp.545-550.
- Kohonen, T. (1988). An introduction to neural computing. *Neural Networks*, Vol.1, pp.3-16.
- Kohonen, T., Barna, G., and Chrisley, R. (1988). Statistical pattern recognition with neural networks: benchmarking studies. *IEEE International Conference on Neural Networks*, San Diego, Ca., July 1988, Vol.1, pp.61-68.
- Lippmann, R.P. (1987). An introduction to computing with neural nets. *IEEE ASSP Magazine*, April, 1987, pp.4-22.
- Manry, M.T. Rotation and scale invariant recognition using a weighted log-polar transform. To be submitted to *Optical Engineering*.
- Powell, M.J.D. (1981). *Approximation theory and methods*. New York: Cambridge Univ. Press.
- Rumelhart, D.E., Hinton, G.E., and Williams, R.J. (1986). Learning internal representations by error propagation. In D.E. Rumelhart and J.L. McClelland (Eds.), *Parallel distributed processing*, Vol.1, Cambridge, Massachusetts: The MIT Press.
- Smith J.L. and D.R. DeVoe (1988). Hybrid optical/electronic pattern recognition with both coherent and noncoherent operations. *Proceedings of SPIE*, Vol.938, pp.170-177.
- Weideman, W.E., Manry, M.T., and Yau, H.C. (1989). A comparison of a nearest neighbor classifier and a neural network for numeric handprint character recognition. *International Joint Conference on Neural Networks*, Washington D.C., June 1989, Vol.1, pp.117-120.

Appendix A. Product Units Versus Min Units

In Kohonen's LVQ network, Min units in the output layer determine the classification result. A Min unit passes the minimum value of its input through to its output. In the NNIN, it is quite possible to use Min units in the output layer. This would result in a network very similar to those of Kohonen. In this appendix, we consider alternative units.

I. Replacements for Min Units

In the NNC, the Min units have two important properties that allow the generation of complicated (disjoint for example) decision regions. These are as follows.

Property 1: Let $\mathbf{r}_{\hat{i}}$ be the vector among the \mathbf{r}_{ij} from which the random vector \mathbf{x} has the smallest distance. If $\mathbf{x} \rightarrow \mathbf{r}_{\hat{i}}$, then $D_{\hat{i}}^2 \rightarrow 0$, and $D_{\hat{i}}^2 = \min D_{ij}^2$ for all $i \in [1, Nc]$, $j \in [1, Ns]$.

Property 2: Property 1 still holds if new reference vectors are added to any class.

Two possible replacements for the Min unit are the product unit (Durbin and Rumelhart 1989) and the sum unit. Define the activation function for the product unit as

$$\text{Pin}(i) = \prod_{j=1}^{Ns} D_{ij}^2 \quad (\text{A1})$$

The product unit has both of the properties given above. As $\mathbf{x} \rightarrow \mathbf{r}_{\hat{i}}$, then $D_{\hat{i}}^2$ and $\text{Pin}(\hat{i})$ both approach 0. Assuming all the \mathbf{r}_{ij} vectors are unequal, then $D_{ij}^2 > 0$ and $\text{Pin}(i) > 0$ for all $i \neq \hat{i}$ and Property 1 is satisfied. When a new reference vector is added, Property 1 and therefore Property 2, are still satisfied. Therefore the product units have the required properties and can function as Min units.

As a second example consider the sum unit, whose activation function is defined as

$$S(i) = \sum_{j=1}^{Ns} D_{ij}^2 \quad (\text{A2})$$

If $\mathbf{x} \rightarrow \mathbf{r}_{\hat{i}}$, then $D_{\hat{i}}$ approaches 0 but $S(i)$ does not. Property 1 is not generally satisfied for the sum unit. If a sum unit did have Property 1 and if a new example vector is added to the \hat{i} th class, $S(\hat{i})$ can no longer be driven to 0 as \mathbf{x} approaches $\mathbf{r}_{\hat{i}}$. Thus the sum unit does not have properties 1 or 2, and is not a suitable replacement for the Min unit.

There are two advantages to using product units rather than Min units; (1) derivatives of products are much simpler to calculate than derivatives of a minimum operation, (2) a back-propagation iteration for product units results in changes for all weights, unlike a similar iteration for min units.

II. Some Properties of Product Units

Assume that $Pw(i,\hat{i}) = \delta(i-\hat{i})$. For each class, put the cluster outputs $Sout(i,j)$ in increasing order such that $Sout(i,1) \leq Sout(i,2) \leq \dots \leq Sout(i,Ns)$. Assume that the correct class is class 1. The Min unit assigns the vector \mathbf{x} to class 1 if

$$Sout(1,1) < Sout(i,1) \quad \text{for } i > 1 \quad (A3)$$

The product unit assigns \mathbf{x} to class 1 if $Pin(1)$ is smaller than $Pin(i)$ for every $i > 1$, which is the same condition as

$$\begin{aligned} Sout(1,1) < Sout(1,1) \left(\prod_{j=2}^{Ns} Sout(i,j) \right) / \left(\prod_{j=2}^{Ns} Sout(1,j) \right) \\ = Sout(i,1) \cdot Rs(i) \end{aligned} \quad (A4)$$

for all $i > 1$. If equation (A3) is true, (A4) is true if either of the following sufficient conditions are true;

- (a) $Sout(1,j) < Sout(i,j)$ for all $i > 1$ and $j \geq 2$
- (b) $Rs(i) < Rs(1)$ for all $i > 1$

Also, if (A4) is true, either of the above conditions is sufficient for (A3) to be true. Thus under certain conditions, product units give the same answer as Min units.

Appendix B. Log-polar feature definitions

Casasent and Psaltis (1977) have proposed log-polar transform (LPT) techniques combining a geometric coordinate transformation with the conventional Fourier transform. Smith and Devoe (1988) have applied the LPT to the magnitude-squared Fourier transform of the image. Lately, some research by Manry has led to a weighted LPT.

The first step in taking the LPT is to Fourier transform the input image $f(x,y)$ to get $F(R,\psi)$, where R and ψ denote frequency domain radius and angle. Next, define $\rho = \ln(R)$ and $L(\rho,\psi) = e^{2\rho} |F(e^\rho, \psi)|^2 = R^2 |F(R,\psi)|^2$. The Fourier transformation of the LPT is taken as

$$M(\zeta,\xi) = \int_{-\infty}^{\infty} \int_0^{\pi} L(\rho,\psi) e^{-j(\zeta\rho + \xi\psi)} d\psi d\rho \quad (\text{B1})$$

The (ζ,ξ) domain is referred to here as the Mellin domain. We define the LPT features g_1 and g_2 as

$$g_1(m) = \frac{|M(m,0)|}{|M(0,0)|} \quad (\text{B2})$$

$$g_2(n) = \frac{|M(0,n)|}{|M(0,0)|} \quad (\text{B3})$$

where $m, n \in [1,8]$.

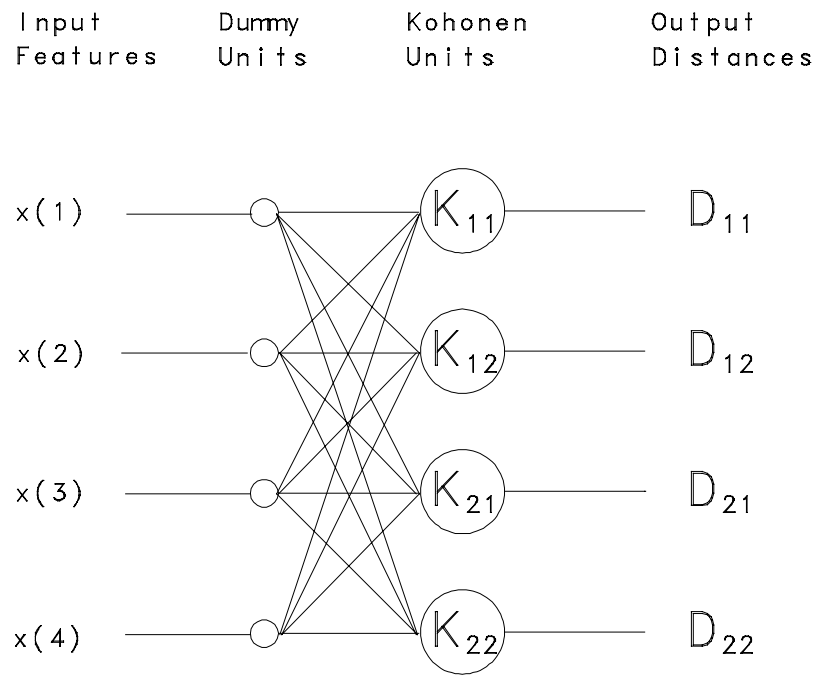


Figure 1. Kohonen Isomorphic Classifier

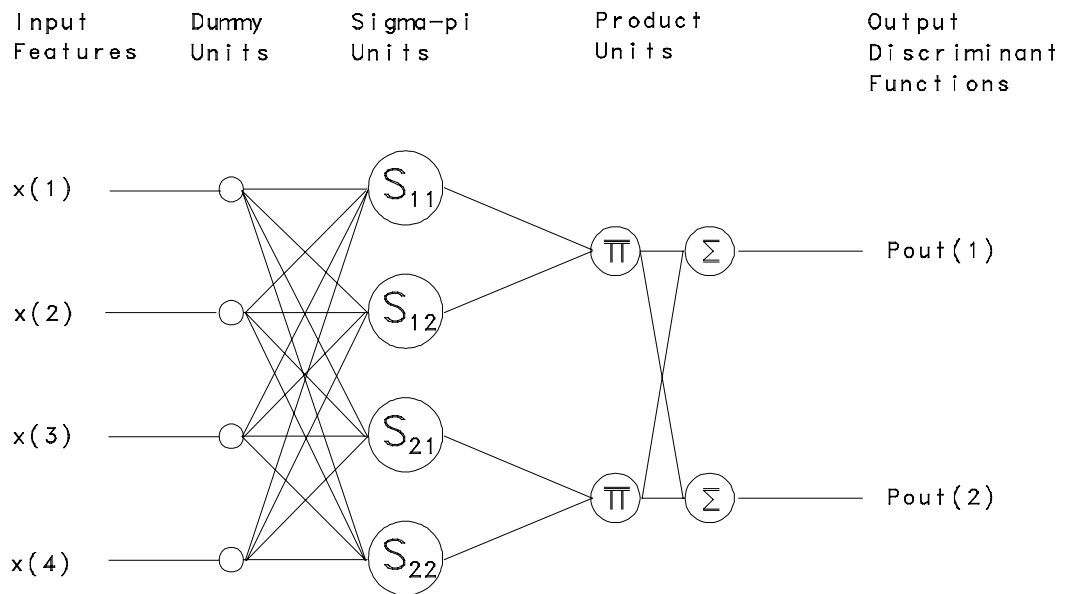


Figure 2. New Isomorphic Classifier

Fig.3 NEURAL NETS IN TRAINING

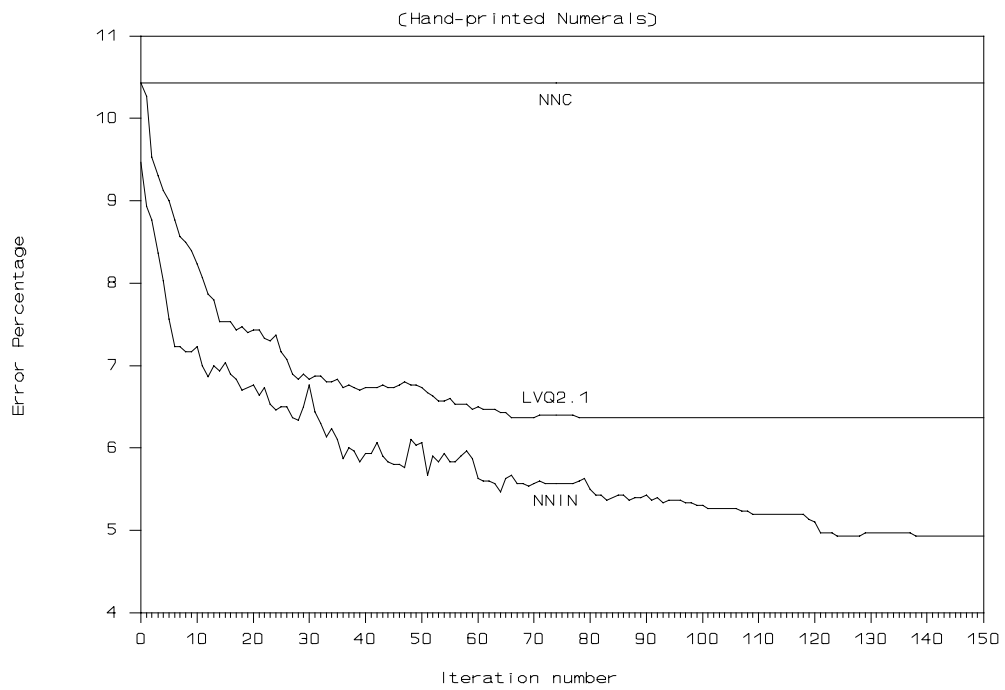


Fig.4 NEURAL NETS IN TRAINING

