Decision-Based Neural Networks with Signal/Image Classification Applications

S. Y. Kung, *Fellow, IEEE*, and J. S. Taur

Abstract—**Supervised learning networks based on a decision-based formulation are explored. More specifically, a decision-based neural network (DBNN) is proposed, which combines the perceptron-like learning rule and hierarchical nonlinear network structure. The decision-based mutual training can be applied to both static and temporal pattern recognition problems. For static pattern recognition, two hierarchical structures are proposed: hidden-node and subcluster structures. The relationships between DBNN's and other models (linear perceptron, piecewise-linear perceptron, LVQ, and PNN) are discussed. As to temporal DBNN's, model-based discriminant functions may be chosen to compensate possible temporal variations, such as waveform warping and alignments. Typical examples include DTW distance, prediction error, or likelihood functions. For classification applications, DBNN's are very effective in computation time and performance. This is confirmed by simulations conducted for several applications, including texture classification, OCR, and ECG analysis.**

I. Introduction

SUPERVISED learning networks represent a mainstream of the development of computational neural networks. Design objectives of these networks must embrace several practical facets, including parallel distributed processing, efficient training and retrieving mechanisms, and training and generalization performances. In supervised training, the training patterns must be provided in terms of input/teacher pattern pairs, denoted as $[\chi, \mathcal{T}] = \{[\boldsymbol{x}_1, \boldsymbol{t}_1], [\boldsymbol{x}_2, \boldsymbol{t}_2], \cdots, [\boldsymbol{x}_M, \boldsymbol{t}_M]\}$, where M is the number of training pairs. The trainee will be told by the teacher what works and what does not, and accordingly makes proper adjustment to improve the performance. Depending on the nature of the teacher's information, supervised learning networks can be divided into two categories: one uses a decision-based formulation and the other an approximation-based formulation.

Approximation-Based Versus Decision-Based Networks: The decision-based and approximation-based formulations differ in their teacher's information and the ways of utilizing it.

1) Approximation-Based Formulation: This formulation is useful when the information from the teacher not only indicates the correctness of the classification but also tells exactly how close the result is to an expected value. This requires the (exact) teacher's values to be available as reference for the output values. The training teacher values $\mathcal{T}$ are real P-dimensional vectors, i.e., $\boldsymbol{t}_i \in R^P$. The objective of network training is to find the optimal weights to minimize the error between the teacher value and the actual response. A popular criterion is the minimum means squares error between the teacher and the actual response. To acquire a more versatile nonlinear approximation capability, multilayer networks (together with the back-propagation learning rule) are usually adopted.

2) Decision-Based Formulation: The DBNN is based on such a formulation. This is a useful formulation when the teacher only tells the correctness of the classification for each training pattern. The teacher is a set of symbols, $\mathcal{T} = \{\boldsymbol{t}_i\}$, labeling the correct class for each input pattern. Unlike the approximation formulation, the exact values of teachers are not required. The objective of the training is to find a set of weights which yields a correct classification. A decision-based learning rule proves to be particularly effective for this purpose.

II. Decision-Based Neural Networks

The fundamental principle of decision-based neural networks has been previously explored and established by many authors, e.g., Fu [5], Kohonen [10], Nilsson [10], and Rosenblatt [17]. The basic configuration of decision-based neural networks is depicted in Fig. 1. A broad variety of decision-based neural networks (DBNN's) has been proposed, ranging from the basic linear perceptron to the more sophisticated hierarchical networks. They all stem from a decision-based learning principle. We shall provide a systematic exploration of several decision-based supervised learning networks. Particularly, a hierarchical DBNN is developed. The new DBNN has three main attributes: perceptron-like learning rule, hierarchical nonlinear network structure, and unification of the static and temporal models.

1) Generalized Perceptron Learning Rule: The reinforced and antireinforced learning rules, originally introduced in the perceptron, have been very appealing from both theoretical and practical perspective. Such a learning rule serves as a guiding design principle of the DBNN. However, the power of a linear perceptron is very limited, since its linear basis function restricts itself to classifications with linear decision boundaries. So it is critical to extend the perceptron by adopting nonlinear discriminant functions to enhance the network's classification power. We propose a gradient updating scheme, that is, the updating direction is taken along the gradients of any general (nonlinear) discriminant function. Such a formulation allows the model to apply not only to static pattern classifications

Manuscript received March 31, 1992; revised March 17, 1993. This research was supported in part by a grant and a research contract from the Air Force Office of Scientific Research and the Defense Advanced Research Project Agency.

The authors are with the Department of Electrical Engineering, Princeton, NJ 08544 USA.

IEEE Log Number 9409819.

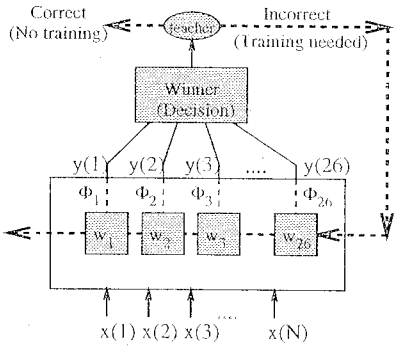


Fig. 1. Decision-based neural network.

but also to temporal pattern classifications. It also extends naturally to cope with pattern classification with fuzzy decision boundary, i.e., classes with overlapping regions. The extension also leads to a minimum error-rate classifier. This will be detailed in Section 3.

2) Hierarchical Nonlinear Network Structure: Very often, the types of nonlinear decision boundaries (and discriminant functions) are too complex to be identified beforehand. To accommodate a versatile nonlinearity, we rely not only on the choice of nonlinear basis functions but also in a sense more on the adoption of a hierarchical structure. For static DBNN's, the nonlinear discriminant function can be best embedded in a hierarchical network structure. A hierarchically structured DBNN offers a great flexibility to accommodate complex decision boundaries.

3) Static and Temporal DBNN's: In designing DBNN's, the most challenging task is to identify a proper discriminant function in order to best distinguish each class from its competing classes. The selection of the discriminant function varies from one application to another. One of the most influential factors hinges upon whether static or temporal patterns are considered.

a) DBNN for Static Pattern Recognition: For static pattern recognition, many basis functions (e.g., linear basis function, radial basis function, and elliptic basis function) may be considered. A hierarchical (hidden-node or subcluster) DBNN's are naturally applicable to static pattern recognitions.

b) DBNN for Temporal Pattern Recognition: For a neural model to fully exploit transient or contextual information, time must play a prominent role. Memoryless networks used for static models are inadequate for temporal pattern recognition. Temporal models must adequately manifest the vital temporal characteristics and remain robust with respect to unpredictable temporal variations such as shift or warping. Fortunately, there are several model-based discriminant functions which are especially suitable and promising for temporal pattern recognition. The temporal functions worthy of consideration include, for example, dynamic time warping (DTW) distance, prediction error, and likelihood functions. More detailed theoretical and application explorations on temporal models can be found in [11].

A. Decision-Based Learning Rules

In training a complex network, the key lies in the following distributive decision-based credit-assignment principles:

1) When to update? In the pure decision-based networks, weight updating is performed only when misclassification occurs. In the fuzzy decision situation, however, a more sophisticated selective training scheme can be utilized to finetune the decision boundaries.

2) What to update? The learning rule is distributive and localized. It applies reinforced learning to the subnet corresponding to the correct class and antireinforced learning to the (unduly) winning subnet.

3) How to update? Because the decision boundary depends on the discriminant function $\phi(\boldsymbol{x}, \boldsymbol{w})$, it is natural to adjust the boundary by adjusting the weight vector $\boldsymbol{w}$ either in the direction of the gradient of the discriminant function (i.e., reinforced learning) or opposite to that direction (i.e., antireinforced learning):

$$\Delta \boldsymbol{w} = \pm \eta \nabla \phi(\boldsymbol{x}, \boldsymbol{w}) \tag{1}$$

where η is a positive learning rate.

The gradient vector of the function ϕ with respect to $\boldsymbol{w}$ is denoted

$$\nabla \phi(\boldsymbol{x}, \boldsymbol{w}) = \frac{\partial \phi(\boldsymbol{x}, \boldsymbol{w})}{\partial \boldsymbol{w}} = \left[\frac{\partial \phi}{\partial \boldsymbol{w}_1}, \frac{\partial \phi}{\partial \boldsymbol{w}_2}, \cdots, \frac{\partial \phi}{\partial \boldsymbol{w}_V} \right]^{\mathrm{T}}$$

where V is the total number of parameters.

Algorithm 1 (Decision-Based Learning Rule): Suppose that $S = \{\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(M)}\}$ is a set of given training patterns, each corresponding to one of the L classes $\{\Omega_i,\ i, \cdots, L\}$. Each class is modeled by a subnet with discriminant functions, say, $\phi(\boldsymbol{x}, \boldsymbol{w}_i)$, $i = 1, \cdots, L$ (cf. Fig. 1.) Suppose that the mth training pattern $\boldsymbol{x}^{(m)}$ is known to belong to class Ω_i; and

$$\phi(\boldsymbol{x}^{(m)}, \boldsymbol{w}_j^{(m)}) > \phi(\boldsymbol{x}^{(m)}, \boldsymbol{w}_l^{(m)}), \qquad \forall l \neq j \tag{2}$$

that is, the winning class for the pattern is the jth class (subnet).

1) When $j = i$, then the pattern $\boldsymbol{x}^{(m)}$ is already correctly classified and no update is needed.

2) When $j \neq i$, that is, $\boldsymbol{x}^{(m)}$ is still misclassified, then he following update is performed.[1]

$$\begin{aligned} &\text{reinforced learning:} && \boldsymbol{w}_i^{(m+1)} = \boldsymbol{w}_i^{(m)} + \eta \nabla \phi(\boldsymbol{x}, \boldsymbol{w}_i) \\ &\text{antireinforced learning:} && \boldsymbol{w}_j^{(m+1)} = \boldsymbol{w}_j^{(m)} - \eta \nabla \phi(\boldsymbol{x}, \boldsymbol{w}_j). \end{aligned} \tag{3}$$

□

Note that, for all $k \neq i$ and $k \neq j$, $\boldsymbol{w}_k^{(m+1)} = \boldsymbol{w}_k^{(m)}$, that is, those weights remain unchanged. The reinforced learning rule moves w_i, along the positive gradient direction; so the value of discriminant function will increase, enhancing the chance of the pattern's future selection. The antireinforced learning rule moves $\boldsymbol{w}_j$ along the negative gradient direction, so the value of discriminant function will decrease, suppressing the chance of its future selection. (Just like the *linear perceptron*, the M training patterns will be repeatedly used for as many sweeps as required for convergence.)

[1]The algorithm remains valid with the following modification: If there is more than one integer j', $j' \neq i$, such that $\phi(\boldsymbol{x}^{(m)}, \boldsymbol{w}_{j'}^{(m)}) > \phi(\boldsymbol{x}^{(m)}, \boldsymbol{w}_i^{(m)})$, then the antireinforced learning can be applied to *all* such indices j'.

Selection of Basis Functions:

1) Linear basis function: In the special linear case, the discriminant function adopted is based on the linear basis function (LBF)

$$\phi(\boldsymbol{x}, \boldsymbol{w}_l) = \boldsymbol{z}^{\mathrm{T}}\boldsymbol{w}_l, \qquad \text{where } \boldsymbol{z} = [\boldsymbol{x}^{\mathrm{T}} \;\; 1]^{\mathrm{T}}.$$

Then the gradient in the updating formula, (1) is simply

$$\frac{\partial \phi}{\partial \boldsymbol{w}} = \boldsymbol{z}$$

which leads to the linear *perceptron* learning rule.

2) Radial basis function: One prominent example is the *LVQ algorithm* proposed by Kohonen [10]. It uses a discriminant function based on the radial basis function (RBF). An RBF discriminant function is a function of the radius between the pattern and a centroid, $\|\boldsymbol{x} - \boldsymbol{w}_l\|$:

$$\phi(\boldsymbol{x}, \boldsymbol{w}_l) = -\frac{\|\boldsymbol{x} - \boldsymbol{w}_l\|^2}{2}, \qquad \text{for each subnet } l \tag{4}$$

is used for each subnet l. So the centroid $(\boldsymbol{w}_l)$ closest to the presented pattern is the winner. By applying the decision-based learning formula to (4) and noting that $\nabla\phi(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{x} - \boldsymbol{w}$, the following leaning rules can be derived:

$$\begin{aligned} &\text{reinforced learning:} && \boldsymbol{w}_i^{(m+1)} = \boldsymbol{w}_i^{(m)} + \eta(\boldsymbol{x} - \boldsymbol{w}_i^{(m)}) \\ &\text{antireinforced learning:} && \boldsymbol{w}_j^{(m+1)} = \boldsymbol{w}_j^{(m)} - \eta(\boldsymbol{x} - \boldsymbol{w}_j^{(m)}). \end{aligned} \tag{5}$$

This is the basic formula for the LVQ (especially LVQ2) algorithm. The RBF decision-based learning is very effective for many practical applications, especially for nearest neighbor-type clustering.

3) Elliptic basis function: The basic RBF version of the DBNN discussed before is based on the assumption that the feature space is uniformly weighted in all the directions. In practice, however, different features may have varying degrees of importance depending on the way they are measured. This leads to the adoption of a (more versatile) elliptic basis function (EBF). The most general form of second-order basis functions is the (skewed) hyperelliptic basis function. In practice and for most applications, the EBF discriminant function is confined to the following (upright) version: The discriminant function (for each subnet l) can be generalized to an (upright) elliptic function:

$$\phi(\boldsymbol{x}, \boldsymbol{w}_l) = \sum_{k=1}^{N} \alpha_{lk}(x_k - w_{lk})^2 + \theta_l \tag{6}$$

where N is the dimension of the input patterns, and $\boldsymbol{w}_l$ is the vector comprising all the weight parameters $\{\alpha_{lk}, w_{lk}, \theta_l\}$. The learning formula can be derived from (3) in Algorithm 1.

A More General Basis Function: In the so-called Φ-function approach in Nilsson [14], the discriminant function for the lth class is

$$\phi_l(x) = \sum_{k_l=1}^{K} w_{k_l}\psi_k(x)$$

where $\psi_k(x)$ are arbitrary linearly independent basis functions. For mathematical rigor, w_{k_l} should be regarded as $w_{k,l}$ both here and in subsequent discussion. The functions $\psi_k(x)$ serve as fixed preprocessing functions so they are not updated. The original input patterns are mapped into a K-dimensional space by the basis functions $\psi_k(x)$'s. The K-dimensional vectors are input to the linear perceptron. The perceptron learning rule is then applied to update w_{k_l}'s. In this way, the nonlinear (e.g. polynomial) discriminant functions can be implemented in the same manner as linear discriminant functions. In terms of the second-order decision boundary, RFB and EBF are just special cases of the Φ function, since they may also be expressed as second-order polynomial basis functions:

$$\phi(x) = w_0 + \sum_{n=1}^{N}(w_{n2}x_n^2 + w_{n1}x_n).$$

Convergence. For the linear perceptron, the convergence may be rigorously established [14]. Similar argument can be applied to the Φ function method, which is an extension of linear perceptron. If the data can be separated by discriminant function ϕ then the new preprocessed data are linearly separable. By applying the linear perceptron convergence theorem, it can be shown that the network will also converge in finite steps [14].

B. Hidden-Node Hierarchical DBNN's

If a subnet is modeled by a single-layer network, it will be inadequate to cope with complex decision boundaries. In order to provide maximum flexibility to accommodate any nonlinear decision boundaries, a hierarchically structured DBNN is proposed. The hierarchical DBNN is characterized by its basis function as well as its hierarchical structure.

Two levels of network hierarchy are being considered:

1) In the conventional single level design, one simple giant network is used for all the cases, i.e., ACON structure, where ACON stands for All-Class-in-One-Net. In the next level of a hierarchical design, the ACON is expanded so that one subnet is devoted to each class. This leads to the "One-Class-One-Net" (OCON) structure. A pioneering example of OCON is the Perceptron Net for the multicategory classification, cf. [2].

2) In the next level, each subnet is modeled based on many subnodes. At this level, either a hidden-node structure or a subcluster structure may be adopted. As an example, the well-known LVQ method in [10] and the piecewise linear perceptron in [14] are actually based on a subcluster structure.

In order to have a consistent indexing scheme for the hierarchical structure, we label the subnet level by the index l and label the subnode level (within a subnet) by the index k_l. More elaborately, the *discriminant function* for subnet l is denoted by $\phi(\boldsymbol{x}, \boldsymbol{w}_l)$. For the lower level, the *local discriminant function* for the subnode k_l is denoted by $\psi_l(\boldsymbol{x}, \boldsymbol{w}_{k_l})$, where the integer $k_l \in 1, \cdots, K_l$, and K_l denotes the number of subnodes in the lth subnet. The most common basis functions, $\psi_1(\boldsymbol{x}, \boldsymbol{w}_{k_l})$, for the subnodes include linear basis functions, radial basis functions, and elliptic basis functions.

In the hidden-node structure in Fig. 2(a), the nonlinear discriminant function is modeled by a weighted sum of several hidden nodes. In this case, a subnet consists of multiple hidden subnodes, each represented by a function $\psi_l(\boldsymbol{x}, \boldsymbol{w}_{k_l})$. The

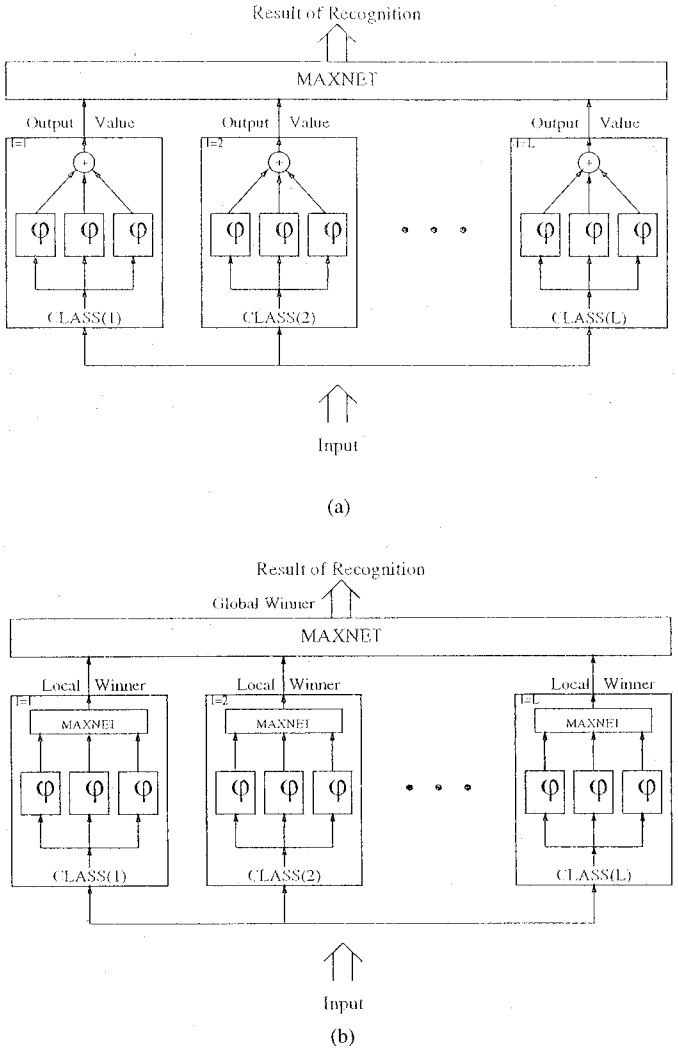


Fig. 2. Schematic diagram of a hierarchical OCON structure. (a) Hidden-node structure. (b) Subcluster structure. In both structures, the higher level consists of multiple subnets (indexed by l), and the lower level consists of multiple subnodes in a subnet (indexed by k_l).

discriminant function of the subnet is a linear combination of the subnode values:

$$\phi(\boldsymbol{x}, \boldsymbol{w}_l) = \sum_{k_l=1}^{K_l} c_{k_l} \psi_l(\boldsymbol{x}, \boldsymbol{w}_{k_l}) \tag{7}$$

where $\{c_{k_l}\}$ denotes the coefficients in the upper layer, and $\boldsymbol{w}_l$ is the vector comprising all the weight parameters. The same decision-based learning rule is adopted (cf. Algorithm 1).

$$\Delta \boldsymbol{w}_l = \pm \eta \nabla \phi(\boldsymbol{x}, \boldsymbol{w}_l).$$

Selection of Neuron Functions: For the hidden-node structure, the net value as expressed by the basis function, $u(\boldsymbol{w}, \boldsymbol{x})$, will be immediately transformed by a nonlinear activation function of the neuron. For example, the most common activation functions are *step, sigmoid*, and *Gaussian functions*. In particular,

1) sigmoid function:

$$f(u_i) = \frac{1}{1 + e^{-u_i/\sigma}}$$

2) Gaussian function:

$$f(u_i) = ce^{-u_i^2/\sigma^2}.$$

For example, the Gaussian RBF is given as

$$\psi_l(\boldsymbol{x}, \boldsymbol{w}_{k_l}) = e^{-\|\boldsymbol{X} - \boldsymbol{W}_{k_l}\|^2/2\sigma_{k_l}^2}. \tag{8}$$

The sigmoid LBF is

$$\psi_l(\boldsymbol{x}, \boldsymbol{w}_{k_l}) = \text{sigmoid}\,(\boldsymbol{z}^{\mathrm{T}} \boldsymbol{w}_{k_l}). \tag{9}$$

Such a discriminant function, with a proper basis function (LBF, RBF, or EBF), can closely approximate any function. It has been shown that any input–output mapping can be approximately realized by various kinds of two-layer networks [1], [6], [15], [16]. This means that the networks may serve as a universal approximator and accommodate almost any complex decision boundaries. Theoretically speaking, the linear-basis DBNN has a universal approximation ability. Moreover, the capacity of the network is not affected by the selection of any (fixed) preprocessing basis functions [14]. Despite this, the basis functions can practically have a major influence on the performance. For example, the radial-basis or elliptic-basis DBNN's have superior performance than linear-basis DBNN's in many practical simulations studied in Section IV.

C. Subcluster Hierarchical DBNN's

In order to further localize the training credit-assignments, a *subcluster structure* is proposed. Instead of using the weighted sum of the node values in the hidden-node structure, the new alternative uses a winner-take-all approach, cf. Fig. 2(b). This is as if only the most representative of the weights in the upper layer has a nonzero weight 1 and all the others have zero weights.

For the subcluster hierarchical structure, we introduce notions of *local winner* and *global winner*. The local winner is the winner among the subnodes within the same subnet. The local winner of the lth subnet is indexed by s_l, that is,

$$s_l = \text{Arg} \max_{k_l} \psi_l(\boldsymbol{x}, \boldsymbol{w}_{k_l}).$$

The global winner is the winner among all the subnets. The jth subnet will be labeled as the global winner if its local winner wins over all the other local winners, that is,

$$\psi_j(\boldsymbol{x}, \boldsymbol{w}_{s_j}) > \psi_l(\boldsymbol{x}, \boldsymbol{w}_{s_l}), \qquad \forall l \neq j.$$

A pattern is classified to the jth class if the jth subnet is the global winner. (Note that, under the subclustering formulation, extra neuron functions at the hidden nodes or the subnet output have no effect on the winner selection.)

The learning rule largely follows Algorithm 1, with the discriminant function of the subnets substituted by that of the local winners:

$$\phi(\boldsymbol{x}, \boldsymbol{w}_i) \Leftrightarrow \psi_i(\boldsymbol{x}, \boldsymbol{w}_{s_i})$$

and

$$\phi(\boldsymbol{x}, \boldsymbol{w}_j) \Leftrightarrow \psi_j(\boldsymbol{x}, \boldsymbol{w}_{s_j}).$$

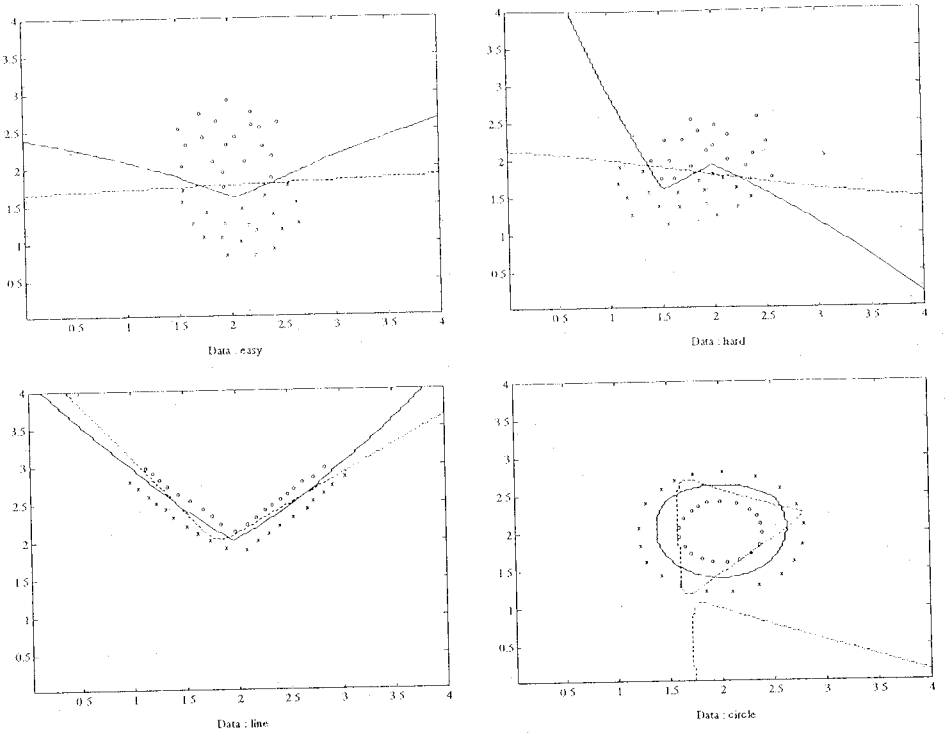


Fig. 3. In this simulation, four different sets of training data are experimented. Each set has two classes (marked by "x" and "o") with 20 patterns in each class. The solid lines and dashed lines are the boundaries generated, respectively, from (EBF) DBNN's and (LBF) approximation-based conjugate-gradient technique [3]. In the retrieving phase of the approximation method, if the output value is greater than 0.5, it is classified into class 1, otherwise class 2. The numbers of hidden units used are 3, 5, 3, and 5 for the four data sets, respectively. The DBNN's successfully classify all the training sets. For the approximation-based net, the success rates are 10/10, 0/10, 2/10, and 6/10, respectively, based on 10 different initial conditions.

Algorithm 2 (Subcluster DBNN's): Suppose that multiple subclusters are used to represent one class, with s_i, s_j, etc., representing the local winners. Suppose that the pattern $\boldsymbol{x}^{(m)}$ should belong to class Ω_i, but the jth subnet is selected as the global winner. When $j \neq i$, that is, $\boldsymbol{x}^{(m)}$ is misclassified, then do the following update:

$$\begin{aligned} &\text{reinforced learning:} && \boldsymbol{w}_{s_i}^{(m+1)} = \boldsymbol{w}_{s_i}^{(m)} + \eta \nabla \psi(\boldsymbol{x}, \boldsymbol{w}_{s_i}) \\ &\text{antireinforced learning:} && \boldsymbol{w}_{s_j}^{(m+1)} = \boldsymbol{w}_{s_j}^{(m)} - \eta \nabla \psi(\boldsymbol{x}, \boldsymbol{w}_{s_j}) \end{aligned} \tag{10}$$

□

In other words, the antireinforced learning is applied to the locally winning subcluster within the globally winning subnet; and the reinforced learning is applied to the local winner within the correct (and supposedly winning) class. Thus, this hierarchical structure can accommodate complex decision boundaries while only the selected subclusters in the subnets are involved in the updating.

Example 1 (Simulation on Two-Dimensional Artificial Data): Four different sets of experiments on two-dimensional artificial data (cf. Fig. 3), involving both the decision-based and approximation-based networks, were performed. The DBNN were found to be trained rapidly, and consistently classify the four sets. In contrast, the approximation-based nets had slower convergence and greater difficulty in forming correct boundaries. The performance of the approximation-based net depended critically on the initial conditions. The success rates for perfect separation of these two classes were, respectively, 100%, 0%, 20%, and 60% for the four experiments in 10 trials. In comparison, DBNN's derived the initial condition via the unsupervised clustering technique, leading eventually to 100% training accuracy for all the four experiments. The decision boundaries created by the two approaches are depicted in Fig. 3.

D. Comparison of DBNN's and Other Models

For static pattern recognition, several basis functions (e.g., LBF, RBF, and EBF) and hierarchical structures are considered. The key variants of static DBNN's are listed in Table I, where the hidden-node DBNN's are denoted DBNN(L_h), DBNN(R_h); and the subcluster DBNNs are DBN(L_s), DBNN(R_s), and DBNN(E_s). Here a capital letter "L", "R", and "E" stands for linear, radial, or elliptic basis functions respectively. LBF or RBF DBNNs offer a much greater classification power compared with the conventional linear perceptron. The RBF DBNNs enjoy a stronger local clustering capability. Based on empirical results, they seem to have slight performance advantage over the LBF DBNNs.

Variants of Hidden-Node DBNN's: In terms of discriminant functions, Φ function [14], potential function [2], probabilistic neural network (PNN) [19], and DBNN(R_h) are very similar to each other. First, they have the discriminant function in the form of a weighted sum of basis functions. Moreover, all of them adopt the OCON structure.

In Duda and Hart [2], a special type of discriminant function for two-class classification is introduced by a notion of potential function. If the patterns are considered as points in the space and each point is associated with a certain amount of electrical charges c_i, then the electric potential function can

TABLE I
KEY VARIANTS OF DBNN'S

	DBNNs	Discriminant/Basis Function	Remarks
Subcluster Hierarchical Structure	DBNN(L_s)	LBF(Linear Basis Function)	Piecewise linear network
	DBNN(R_s)	RBF (Radial Basis Function)	· Also named GLVQ
	DBNN(E_s)	EBF (Elliptic Basis Function)	· Generalization of LVQ
Hidden-Node Hierarchical Structure	DBNN(L_h)	Weighted Sum of Sigmoid LBF	· Use BP algorithm
	DBNN(R_h)	Weighted Sum of Gaussian RBF	· Generalization of PNN

be written in the following form:

$$\phi(x) = \sum_{i=1}^{M} c_i \psi(x, x_i)$$

where M is number of patterns and $\psi(x, x_i)$ is the potential function due to the unit charge at position x_i. Patterns in the same class have the same sign of the electric charges. To achieve a better flexibility, alternative potential functions other than the physical electric potential function are often adopted. For example, the Parzen window estimate [2] and the PN are special cases. In PNN, the Gaussian radial basis function $\exp(|\boldsymbol{x} - \boldsymbol{x}_i|^2/\sigma)$ is used as the potential function. In the DBNN's, one centroid (parameterized by w_i) is used to represent multiple patterns (x_i's) in the same cluster. Thus, they are more flexible and yet use less parameters. However, it also brings about the need of training centroids. More precisely, the $\psi(x, w_{k_l})$ functions in (7) are parameterized with w_{k_l} and updated according to the reinforce and antireinforce learning rule. In a later variant of PNN, the number of subnodes is reduced, making it closer to the radial-basis DBNN(R_h). For the PNN, the centroids w_i are computed from a clustering scheme and there is no further training involved after the centroids are obtained.

Hybrid Training Rule for Potential Function: The updating rule for the potential function, as introduced in [2], makes use of both the hidden node and the subcluster techniques. In the retrieving phase, the $\phi(x)$ is used as a discriminant function for a two-class classification, just like the hidden-node structure. (A pattern is classified into class 1 if $\phi(x) > 0$, class 2 otherwise). In the learning phase, the updating formula, however, follows that used in the subcluster structure with $\psi_i(x) = c_i K(x, x_i)$ where the c_is are the only parameters to be updated. Suppose that the ith pattern (x_i) is being considered. If x_i should belong to class 1 but is misclassified into class 2, then the parameter c_i is updated according to the reinforced learning rule:

$$c_i = c_i + \eta$$

where η denotes a positive learning rate. If x_i should belong to class 2 but is misclassified into class 1, then the antireinforced learning

$$c_i = c_i - \eta$$

is performed.

Variants of Subcluster DBNN's:

1) Piecewise Linear Network Versus DBNN: The notion of hierarchical network has existed for several decades [2], [14], [10]. In particular, the piecewise linear machine proposed in [14] has the same structure as the subcluster DBNN's, except that the discriminant function is limited to linear basis function. Under the LBF formulation, it may be appreciated by the classification power of piecewise linear decision boundary with each linear segment determined by the responsible local winners.

2) LVQ Versus DBNN: The DBNN's with radial basis and elliptic basis functions, DBNN(R_s) and DBNN(E_s) have a strong resemblance to the well-known LVQ algorithm; cf. (5). The subcluster RBF DBNN (DBNN(R_s)) is closest to LVQ, especially LVQ2. They share many common attributes:

1) They both adopt reinforced/antireinforced learning rule.
2) They both adopt a radial basis subcluster scheme. Thus the amount of update is proportional to the distance between the pattern and the centroid; cf. (5). (Sometimes it is useful to further extend the radial basis DBNN to the elliptic basis version.)

The updating subclusters are generally not the same for the (RBF) DBNN and LVQ2. The LVQ2 updates the local winner of the globally winning class. If the runner-up class is the correct one, then the local winner of the runner-up class is also updated. The DBNN updates the local winner of the globally winning class and the local winner of the correct (and supposedly winning) class. However, the correct class may not necessarily be selected as the runner-up class; therefore, the credit-assignment schemes of DBNN and LVQ2 are not the same.

Subcluster Versus Hidden-Node Structures: To relate the subcluster and hidden-node approaches, it is useful to adopt a Gaussian RBF and to use a distribution function perspective. The subcluster technique is based on winner-take-all with one (winning) centroid selected as the only representative centroid for the pattern. The hidden-node technique, on the other hand, adopts a distributed approach that each of the centroids is given their fair share of representation (denoted by the coefficients c_i).

The winner-take-all approach has the advantage that it is simpler and more effective to implement and it can converge to a good result when the training patterns are all very clean and consistent. In contrast, the distributed hidden-node technique is more robust. It is more costly to implement. However, it

offers robustness when the training patterns (as well as the testing patterns) are subject to measurement or other noises. (The notion of fuzzy decision in Section IIIB may provide a complementary solution.)

Under a local receptive field, such as the Gaussian RBF, the hidden-node and subcluster DBNN's may exhibit a very similar behavior. Since the magnitude of a Gaussian RBF $\psi_i(\boldsymbol{x}, \boldsymbol{w}_i)$ drops very quickly when the pattern is far away from the centroid $\boldsymbol{w}_i$, the closest subnode will easily dominate its share in the weighted sum and the far away subnodes have very little effect on the value of ϕ. In this sense, the difference between the winner-take-all and the distributed techniques may be fairly minor.

TABLE II
COMPARISON OF VARIOUS DBNN'S FOR THE TEXTURE-CLASSIFICATION

Network	Noise Tolerance	Test Set
DBNN(L_h)(20)	0(200 sweeps)	3.25%
DBNN(R_s)(4)	0(20 sweeps)	2.88%
DBNN(R_s)(4)	0.13	2.42%
DBNN(R_s)(8)	0(7 sweeps)	3.54%
DBNN(R_s)(8)	0.2	2.92%
DBNN(E_s)(1)	0(200 sweeps)	3.04%
DBNN(E_s)(4)	0(17 sweeps)	2.46%
DBNN(E_s)(4)	0.15	2.08%

III. PRACTICAL EXTENSIONS OF DBNN'S

The network capability/capacity concerns the existence of a weight solution so that the network can deliver a preset classification and/or approximation performance. Both the *memorization* and *generalization* share the same objective of training a network so that it may accurately classify the given test patterns. However, the nature of the test patterns can make a good deal of difference. For memorization, the test pattern is one of the original training patterns. For generalization, however, the test pattern may not necessarily be from the original training patterns. At the best, the test pattern is drawn from the same distribution which generates the training patterns in the first place. This difference has a major influence on the problem formulation. While a good classification/approximation performance during the training phase almost invariably means a good memorization capability, it does not necessarily guarantee a good generalization capability.

As to numerical efficiency in search of the weights, the (back-propagation) numerical method still encounters very serious difficulties. Therefore, the ease of learning constitutes a critical criterion, and many practical factors, e.g., convergency, computational cost, and local minimum, must be taken into account. In this section, we should first look into some critical design factors regarding generalization capability. Then a fuzzy decision training scheme will be proposed as an alternative approach.

A. Training and Generalization Performance

1) Number of Subnodes: Several approaches can be used to estimate the number of hidden nodes or subclusters. It can be either predetermined based on, for example, some prior knowledge on the training pattern distribution. It also can be determined based on the clustering technique adopted. There exists a tradeoff between the training accuracy and the generalization performance (during the test phase). More precisely, when too many subclusters or too many hidden nodes are adopted, the improved approximation is often a result of overtraining (i.e., overfitting). This in turn will hamper the model's generalization capability.

Simulation results show that (cf. Table II) a reduced number of hidden-nodes or subclusters can improve generalization (sometimes at the expense of training accuracy). AIC type of criterion [12] may be adopted to determine the optimal number of subnodes. However, some modification of AIC is necessary in order to cope with the mutual training strategy used here. This subject is currently under investigation.

2) Convergence: A remark about the convergence is in order here. There is no rigorous proof of convergence for hierarchical DBNN's. For patterns with abnormal distributions, ad-hoc rules are needed for the termination of the training process. Nevertheless, with a proper choice of discriminant functions, rapid convergence has been observed in practice. Furthermore, the DBNN's have demonstrated high performance and effectiveness in many practical applications, cf. Section IV.

3) Unsupervised VQ Method for Initial Weights: A good initial condition is critical to convergence. In our simulation, K-means clustering algorithm [2] is used to obtain the initial values for static discriminant functions. Based on this initial condition, the decision-based learning rule can be applied to further fine tune the decision boundaries. This scheme is used in the subsequent DBNN simulations. For temporal classification, the initial values are independently trained by optimizing the individual discriminant function.

B. Fuzzy Decision Neural Networks

When pattern categories are clearly separated, there is a range of feasible weight solutions. This fact can be exploited to attain the widest possible margin of separation (denoted as ϵ) between two neighboring regions. This further leads to a modified learning rule, nearly the same as Algorithm 2.1, with the exception that (2) is substituted by

$$\phi(\boldsymbol{x}^{(m)}, \boldsymbol{w}_j^{(m)}) > \phi(\boldsymbol{x}^{(m)}, \boldsymbol{w}_l^{(m)}) + \epsilon. \tag{11}$$

Fortification by such a positive *vigilance* ϵ usually yields a better generalization. However, when classes are not clearly separable, a very different approach will be required to cope with patterns in the border area. By properly imposing vigilance and/or tolerance in the training phase, a better generalization can be attained.

To illustrate the need of a fuzzy decision, let us study distribution functions for the following two classes of artificial

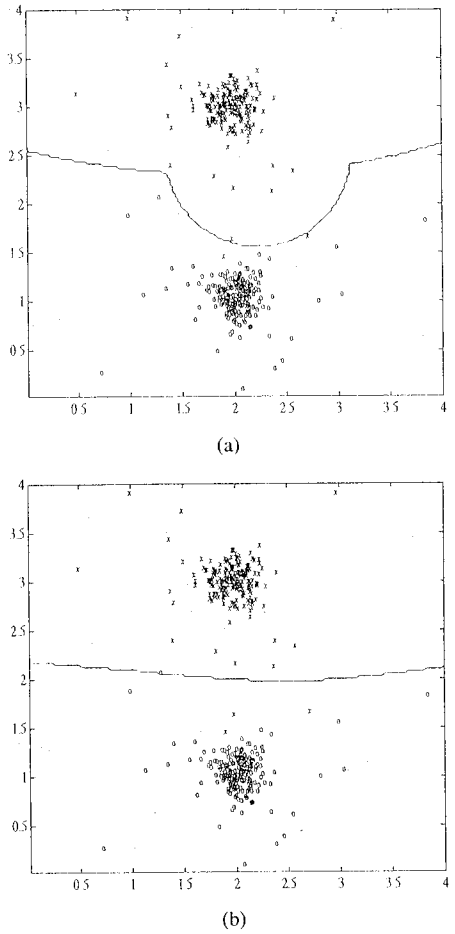


Fig. 4. Patterns from two nonseparable classes are labeled with x's and o's. (a) The decision boundary obtained by DBN(E_s), with two subclusters. (b) The decision boundary by FDNN(E_s).

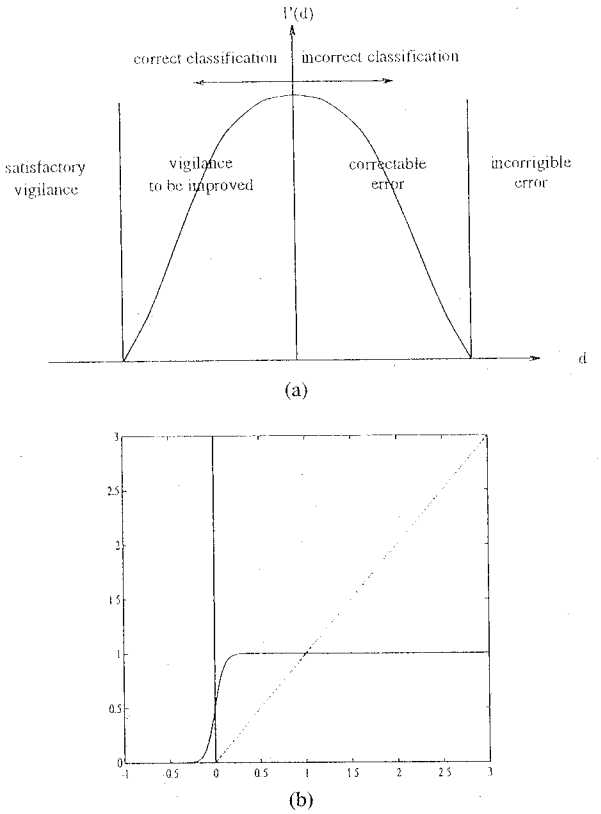


Fig. 5. (a) For DBNN, only two regions are of interest: misclassification versus correct classification. In contrast, for FDNN, four regions are distinguished: 1) correct with satisfactory vigilance, 2) correct with vigilance to be improved, 3) error on which correction will be attempted, and 4) error to remain uncorrected. (b) The difference between the penalty functions of a hard-decision DBNN (linear line) and a fuzzy-decision neural network.

patterns:

$$p_1(x, y) = 0.9 \times N(x, 3, s^2)N(y, 2, s^2) + 0.1 \times N(x, 3, 5 * s^2)N(y, 2, 5 * s^2) \quad (12)$$

and

$$p_2(x, y) = 0.9 \times N(x, 1, s^2)N(y, 2, s^2) + 0.1 \times N(x, 1, 5 * s^2)N(y, 2, 5 * s^2) \quad (13)$$

where $N(t, \mu, s^2)$ is Gaussian with mean μ and variance s^2. The training patterns contain additive noise, represented by the second Gaussian term in the distribution in (12) and (13). Due to the noise, the two classes overlap in the feature space, as shown in Fig. 4. (Here, we set $s = 0.15 \ldots$) If a (hard) DBNN is used for the two classes, all the patterns are accounted for in the training process. An almost error-free training accuracy can be attained after iteratively adjusting the boundary. Depicted in Fig. 4(a) is a very twisted decision boundary obtained by the DBNN. In order to achieve a better generalization, we need to incorporate some *tolerance* of misclassification.

One way of providing tolerance, for the nonseparable case, is simply to ignore the persistently undecided patterns. A more elegant and suitable approach can be derived based on a somewhat fuzzy decision, leading to the fuzzy decision neural network (FDNN) [8], [20]. More precisely, as depicted in Fig. 5(a), there are different degrees of error associated with each decision, for example, marginally erroneous, erroneous, and extremely erroneous. The technique imposes a proper penalty function on all the "bad" decisions as well as the "marginally correct" ones. The final solution represents the best compromise in terms of the total penalty. In short, this allows "soft" or "fuzzy" decision, as opposed to the hard (yes or no) decision. To cope with "marginal" training patterns, and to provide a smooth "gradient" for learning, the penalty must be a function of the degree of error. Fuzzy decision neural networks have the following special features: 1) good classification performance with large overlapping between classes, 2) minimum error classifier, and 3) selective training scheme.

Suppose that $S = \{\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(M)}\}$ is a set of given training patterns; and the discriminant function for the class Ω_i is denoted $\phi(\boldsymbol{x}, \boldsymbol{w}_i)$, for $i, \cdots, L$. For DBNN, the *winner* class is denoted as Ω_j where $j = \arg\max_j \phi(\boldsymbol{x}^{(m)}, \boldsymbol{w}_j)$. However, for FDNN, an alternative denotation is adopted. Instead, Ω_j now denotes the leading *challenger* among all the classes *excluding* the correct class Ω_i. That is,

$$j = \arg\max_{j \neq i} \phi(\boldsymbol{x}^{(m)}, \boldsymbol{w}_j). \quad (14)$$

For a training pattern, a measure of misclassification can be

introduced:

$$d = d^{(m)}(\boldsymbol{x}^{(m)}, \boldsymbol{w}) = -\phi_i(\boldsymbol{x}^{(m)}, \boldsymbol{w}_i) + \phi_j(\boldsymbol{x}^{(m)}, \boldsymbol{w}_j) \quad (15)$$

where $\boldsymbol{w}$ denotes all the involved weight vectors. In fact, a more general measure was proposed in [7]–[9] as

$$d = -\phi_i(\boldsymbol{x}^{(m)}, \boldsymbol{w}_i) + \left\{ \frac{1}{L-1} \sum_{j,\, j\neq i} \phi_j(\boldsymbol{x}^{(m)}, \boldsymbol{w}_j)^{\gamma} \right\}^{1/\gamma}, \quad \gamma > 0 \quad (16)$$

where ϕ is assumed to be positive. Note that, when $\gamma \to \infty$, (16) will closely approximate (15).

Two scenarios are of interest. 1) When d is positive, then the associated pattern would be *misclassified* to the challenger Ω_j. In this case, just like DBNN, the updating process is desirable. In fact, the larger the magnitude of d, the greater the error. However, if d is too large, the training pattern is probably corrupted by large noise and should be ignored. 2) When d is negative, then the pattern can be correctly classified to Ω_i. Nevertheless, if d is only a small negative value, then the correct choice Ω_i would win over the challenger Ω_j only by a narrow margin. In this case, unlike DBNN, it may still be advisable to invoke the learning process so as to enhance the vigilance. A better vigilance margin is guaranteed as the magnitude of d gets larger. This suggests that a window can be used to mask out the interval for d in which updating is desired (cf., Fig. 5(a)).

Penalty functions which lead asymptotically to a minimum error classification (cf. Fig. 5(b) solid curve) are of most interest. For example,

$$l(d) = \frac{1}{1 + e^{-d/\xi}} \quad (17)$$

$$l(d) = \begin{cases} 0 & d \leq -\xi/2 \\ (d + \xi/2)/\xi & -\xi/2 < d < \xi/2 \\ 1 & \xi/2 \geq d. \end{cases} \quad (18)$$

In particular, when the parameter ξ approaches zero, these penalty functions approach the step function

$$l(d) = \begin{cases} 0, & d < 0 \\ 1, & d \geq 0. \end{cases} \quad (19)$$

Note that a complete denotation for the penalty function for an individual training pattern $\boldsymbol{x}^{(m)}$ should have been $l(\boldsymbol{x}^{(m)}, \boldsymbol{w}) = l(d(\boldsymbol{x}^{(m)}, \boldsymbol{w}))$; and the overall cost function is $E(\boldsymbol{w}) = \sum_{m=1}^{M} \{l(d^{(m)})\}$. If the step penalty function in (19) is adopted, this cost function would be the same as the recognition error count. Even with the differentable forms of (17) or (18), the cost function still represents a good approximation of the total number of errors. Fig. 5(b) compares the soft penalty function, such as (17), with the corresponding penalty function used in the DBNN:

$$l(d) = \zeta d. \quad (20)$$

Note that the linear penalty function imposes too excessive a penalty for patterns with large margins of error; thus the sigmoid function is more appealing. It effectively treats the errors with equal penalty once the magnitude of error exceeds a certain threshold. Consequently, these errors may be given up for the interest of the majority of training patterns. This strategy is in agreement with the so-called minimum error rate classifier defined by Duda [2]. Furthermore, now the gradient descent method can be applied to minimize the overall cost function. The soft penalty function provides a means to minimize the number of recognition errors, at least approximately.

Algorithm 2 (Fuzzy Decision Learning Rule): Suppose that the mth training pattern $\boldsymbol{x}^{(m)}$ is known to belong to class Ω_i; and that the leading challenger is denoted $j = \arg\max_{j\neq i} \phi(\boldsymbol{x}^{(m)}, \boldsymbol{w}_j)$. The learning rule is

reinforced learning:

$$\boldsymbol{w}_i^{(m+1)} = \boldsymbol{w}_i^{(m)} + \eta l'(d) \nabla \phi(\boldsymbol{x}^{(m)}, \boldsymbol{w}_i)$$

antireinforced learning:

$$\boldsymbol{w}_j^{(m+1)} = \boldsymbol{w}_j^{(m)} - \eta l'(d) \nabla \phi(\boldsymbol{x}^{(m)}, \boldsymbol{w}_j)$$

where $l'(d)$ is the derivative of the penalty function evaluated at d and it controls the size and shape of the updating window (cf. Fig. 5(a).) □

Simulation results confirm that the FDNN works more effectively than the DBNN when the training patterns are not separable. Indeed, the FDNN yields a very smoother decision boundary for the aforementioned Gaussian-mixture problem; cf. Fig. 4(b).

Minimum Error and Unbiased Classifier: Theoretically speaking, the DBNN is biased, while the FDNN is less biased. If the window size ξ is large, however, the FDNN will be more prone to bias [20]. Therefore, a learning method with adjustable window size, such as that used in simulated annealing, may be considered. More precisely, in the beginning phase, the window size should be large enough so that the decision boundary can move rapidly to a correct neighborhood. In order to reduce bias, the window size should gradually decrease. Under idealistic circumstances, the minimum error rate can be yielded.

Selective Training: In DBNN's, only the misclassified patterns are involved in the updating. For FDNN, this principle has to be modified due to the fuzziness of decision boundary. If the loss function is defined as in (17), then the parameters in the corresponding subnets will be updated for each pattern. In this case, the computational cost will be higher. However, if (18) is used, only the patterns which fall into a pre-specified rectangular window will require updating. Most of the correct and safe patterns are no longer involved in the updating in the final stage of training FDNN. They may be waived from the future training process, if they are correctly classified with a respectable safety margin for a sufficient period during the training process.

IV. Signal/Image Classification Examples

The DBNN and FDNN formulation can be effectively blended into traditional recognizers to deal with both static and temporal recognition problems. The possible discriminant functions for the static problems include different basis

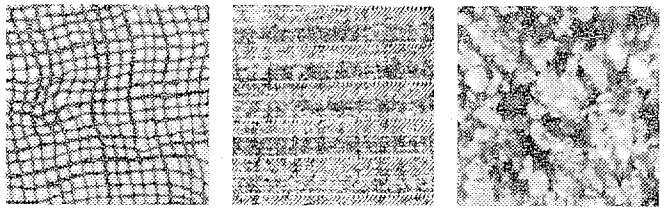


Fig. 6. Sample textures used in the texture-classification experiments.

functions combined with hierarchical structure. Examples of temporal discriminant functions are likelihood function, prediction error, and DTW distance. In order to demonstrate the applicability of DBNN's and FDNN's, two static recognition examples (OCR and texture classification) and two temporal examples (ECG recognition and DTW network) are studied.

A. Texture Classification

The DBNN's perform very well in texture-classification applications. Some texture samples are given in Fig. 6. The performances of several DBNN's, including RBF and EBF subclustering DBNN's (DBNN(R_s) and DBNN(E_s)) and an LBF hidden-node DBNN (DBNN(L_h)) are compared in the study.

Feature Extraction: The texture feature used here is based on a compressed representation of the *texture spectrum* originally proposed by [22]. The texture vector associated with a pixel is characterized by 8 ternary values {0, 1, or 2}, labeling the relative level between the central pixel and its 8 immediate neighbors. More precisely, if a neighbor pixel level is relatively lower (equal, or higher, respectively), then its corresponding value will be labeled 1 (0, or 2, respectively). For each central pixel, the total number of possible *texture vectors* is 3^8; cf. Fig. 7. Because the input dimension (3^8) is too large, we adopt a simplified representation in order to save computation time and storage. It is based on three integers $\{x, y, z\}$ defined as the numbers of 0's, 1's, and 2's in a texture vector, respectively. With the new representation, there are now only 45 possible combinations of $\{x, y, z\}$. Thus the reduced spectrum with dimension 45 can be obtained by calculating the histogram of such representation of the texture vectors in that block; cf. Fig. 8. The texture vector (and thus the texture spectrum) contains the information of the local texture structure of the image. One advantage of the texture spectrum lies in its insensitivity to the variation of the background intensity and noise because only the relative grey level is relevant.

In the simulation study, a total of 12 Brodatz textures (texture numbers 3, 16, 28, 33, 34, 49, 57, 68, 77, 84, 93, and 103) are used. For each texture image, 529 32×32 blocks are sampled uniformly across the entire image. Their reduced spectra are then computed which in turn are used as the training data. By a similar method, additional 200 blocks are randomly chosen from the same texture image to form the test set. The linear-basis hidden-node structure, DBNN(L_h), and two subcluster structures, DBNN(R_s) and DBNN(E_s), have been tried. The classification performance is summarized in Table II, showing very satisfactory convergence speeds and

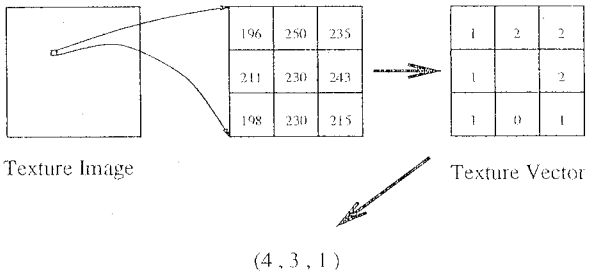


Fig. 7. Example of texture vector and its simplified representation. (Note that there are four 1's, one 0, and three 2's in the texture vector.)

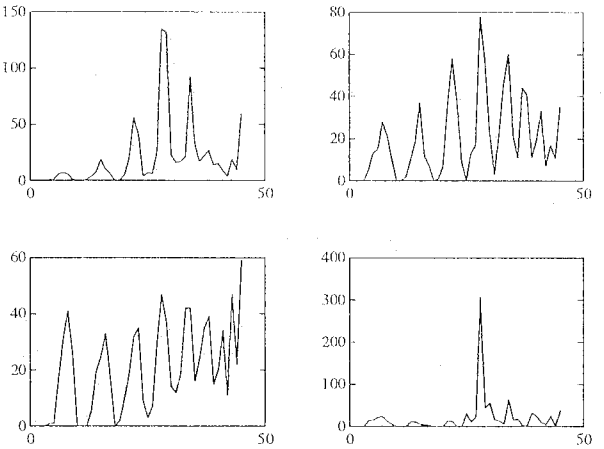


Fig. 8. Sample reduced texture spectrum from different classes.

classification accuracies. The generalization performance of DBNN(E_s) is slightly better than that of DBNN(R_s), with the DBNN(L_h) as a distant third. Additional experiments indicate that the linear perceptron fails to separate the 12 classes and the approximation-based BP method has very slow speed and persistently large mean squares error (after 500 sweeps).

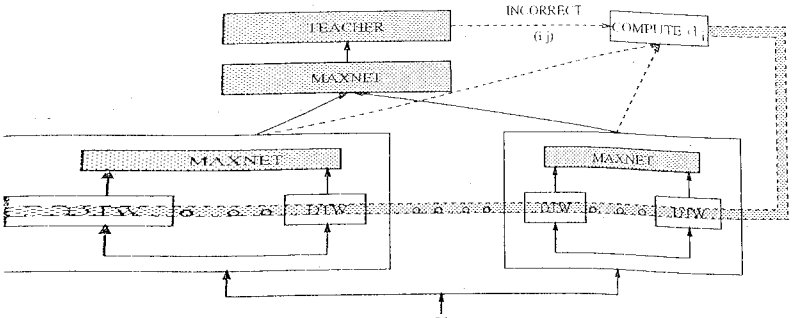
B. Optical Character Recognition (OCR) Application

The problem is to recognize a rectangular pixel array as one of the 26 capital letters in the English alphabet. The character images were based on 20 different fonts, and each letter within these 20 fonts was randomly distorted to produce a file of 20 000 stimuli. There are three types of distortion including linear magnification, aspect ratio, and horizontal and vertical warping. Each character was converted into 16 primitive numerical attributes (in terms of statistical moments and edge counts) that were then scaled to fit into a range of integer values from 0 through 15. (The source file of the letter image recognition data was generated by Slate, cf. [4]). The first 16 000 patterns are used as the training patterns and the remaining 4000 serve as the test pattern.

Several Holland-style adaptive classifiers were experimented in [4], obtaining an accuracy of a little over 80%. On the other hand, the simulation results on the DBNN's, summarized in Table III, are considerably better. In this simulation, 10-subcluster DBNN(E_s) performs better than 20-subcluster DBNN(R_s). (Note that the two nets have approximately the same number of weight parameters.) Both

TABLE III
COMPARISON OF TWO DBNN'S FOR THE OCR CLASSIFICATION

>rithm	No. of Clusters	Training Set	Test Set
N(E_s)	10	0.0375%	6.2%
N(R_s)	20	0.025%	7.5%



Possible DTW FDNN structure for speech recognition.

's yield better performance than the results reported

Temporal Networks for Speech Recognition

>h recognition represents another important potential ion of neural networks. The dynamic time warping technique [18], [13] is a well-known conventional h for speech recognition. It is natural to explore some ation between DBNN (FDNN) and DTW for speech or mporal signal recognition.

apporal speech vector can be formed by, for example, :tral coefficients of the speech waveforms. Both the e signal w and the training signal x are represented quence of vectors. It is practical to assume that both ing and test signals are inherently warped. So the best e" between the two sequences cannot be derived from comparison of the two vectors based on the original exing (i.e., template matching). Instead, the distance e based on a new indexing which takes the warping to account. The DTW distance measure provides tive means to cope with such a time variability The time-warping factor is carefully decoded during rocessing phase.

bjective of the DTW is to find a scheme to reindex ence and training signal sequences so that the best tween two sequences can be attained [18]. The total the warping path can be used as a discriminant $\phi(x, w)$. The mutual training incorporated into the ision-based neural network should help enhance the performance (cf. Fig. 9).

ral DBNN Models for ECG Analysis

diction-based discriminant function is proposed for pattern recognition, and particularly, for ECG recog- . The prediction-based classifier has the same theo- is as the linear predictive classifier (LPC). According ON structure, one temporal net is assigned to each class. A long sequence x wth length $\overline{N}$ is input through a tapped-delay line, from which a set of N-dimensional vectors ($N \ll \overline{N}$), denoted as $x_j = [x(j+1) \cdots x(j+N)]$, are consecutively extracted. Here $x(n)$ denotes the nth sample value of the signal sequence x. These N-dimensional vectors serve as the inputs to the back-propagation network. The function is denoted as $f_a(x_j) = f(x_j, w)$, which could be, for example, either a sigmoid LBF or a Gaussian RBF.

For each training pattern, the discriminant function ϕ [cf. Fig. 1) is defined as the (negative) squared prediction error,

$$\phi(x, w) = -\sum_{j=0}^{\overline{N}-N-1} (f_a(x_j, w) - x(j+N+1))^2.$$

The design hierarchy can be divided into two phases. In the *independent training* phase, each model is trained by patterns from its own category. The objective is to minimize the sum of the squared prediction errors over all the M training patterns in the same class. This is very appealing to temporal pattern recognition and offers cost-effective learning in the initial phase. If further training is needed, a DBNN-like *mutual training* strategy can always follow. Either DBNN or FDNN may be adopted to fine-tune the $\phi(x, w)$. In the *retrieving phase*, just like the DBNN, a pattern is classified into the subnet which yields the largest discriminant function, (i.e., the smallest prediction error).

In the ECG case study, there are 10 different ECG classes, each having 10 sample waveforms. By waveform time-shift, the original data set is expanded to a total of 500 patterns. The models were trained by the original data but tested (mostly) by the shifted data. (More precisely, 50 original waveforms are used as the training set and the remaining 450 are test patterns.)

Both static and temporal networks are evaluated. Their generalization performance and tolerance to time misalignments are studied. For the static models, a subclustering hierarchical DBNN is adopted. For the temporal prediction-based models, in order to smooth the noise in the signal, a fixed-weight lowpass filter is used to process the teacher value for the prediction network. The experimental results suggest that the temporal nets are inherently tolerant to temporal misalignment of waveforms. While the static DBNN(E_s) yields only around 85% testing accuracy the (LBF and/or RBF) "prediction-based" temporal models can achieve better than 98.45% accuracy. (For comparison, the conventional linear predictive classifier has a performance around 90%). For more details, see [21].

V. CONCLUDING REMARKS

The main thesis of this paper is that, for classification applications, the decision-based approach is more natural and in general performs better than the approximation-based approach. The first DBNN-type neural model ever introduced is the linear *perceptron* which adopts a modest single-layer structure with a linear basis function. Multilayer and hierarchical network structures with nonlinear basis functions are more appealing to a broader application domain. Accordingly, DBNN's have a modular and hierarchical architecture.

More importantly, it adopts a competitive credit-assignment scheme that decides which subnets and/or subnodes should be trained or used. The hierarchical structure provides a unified framework for other better known models (e.g., perceptron and LVQ), and offers a better understanding of the structural richness of decision-based neural networks. This structure can also be embedded naturally in fuzzy-decision neural networks (FDNN's). Both DBNN's and FDNN's can be applied to temporal pattern recognition problems by adopting a proper discriminant function. Based on simulation performance comparison, the DBNN's appear to be very effective for many signal/image-classification applications.

REFERENCES

[1] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, pp. 303–314, 1989.

[2] R. O. Duda and P. E. Hart, "Pattern Classification and Scene Analysis. New York: Wiley, 1973.

[3] B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C*. W. H. Press, 1990.

[4] P. W. Frey and D. J. Slate, "Letter recognition using Holland-style adaptive classifier," *Machine Learning*, vol. 6, no. 2, pp. 161–182, Mar. 1991.

[5] K. S. Fu, *Sequential Methods in Pattern Recognition and Machine Learning* Orlando, FL: Academic Press, 1968.

[6] K. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, pp. 183–192, 1989.

[7] B. H. Juang and S. Katagiri, "Discriminative learning for minimum error classification," *IEEE Trans. Signal Processing*, in press.

[8] S. Katagiri, C. H. Lee, and B. H. Juang, A generalized probabilistic decent method. In *Proc. Acoust. Soc. of Japan*, pp. 141–142, Nagoya, Sept. 1990.

[9] ———, "Discriminative multi-layer feed-forward network." In B. H. Juang, S. Y. Kung, and C. A. Kamm (Eds.), *Neural Networks for Signal Processing*, I. Proc. of the 1991 IEEE Workshop, Princeton, NJ, 1991.

[10] T. Kohonen, *Self-Organization and Associative Memory*, Series in Information Science, vol. 8, Springer-Verlag, New York, 1984.

[11] S. Y. Kung, *Digital Neural Networks*. Englewood Cliffs, NJ: Prentice Hall, 1993.

[12] L. Ljung, *System Identification: Theory for the User*. Englewood Cliffs, NJ: Prentice Hall, 1987.

[13] C. Myers, L. R. Rabiner, and A. E. Rosenberg, "Performance tradeoffs in dynamic time warping algorithms for isolated word recognition," *IEEE Trans. Acoustics, Speech, and Signal Proc.*, Vol. ASSP-28 (No. 6):623–635, Dec. 1980.

[14] N. J. Nilsson, *Learning Machines*. New York: McGraw-Hill, 1965.

[15] E. Parzen, "On estimation of a probability density function and mode," *Anal. Math. Stat.*, vol. 33, pp. 1065–1076, 1962.

[16] T. Poggio and F. Girosi, "A theory of networks for approximation and learning," Tech. Rep. AI Memo 1140, 1989.

[17] F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Washington, DC: Spartan, 1961.

[18] H. Sakoe and S. Chiba, "Dynamic programming optimization for spoken word recognition," *IEEE Trans. Acoustics, Speech, Signal Proc.*, vol. 26, pp. 43–49, Aug. 1978.

[19] D. F. Specht, "Probabilistic neural networks," *Neural Networks*, vol. 3, pp. 109–118, 1990.

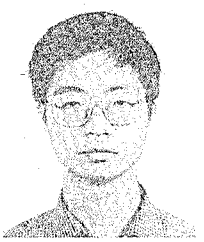
[20] J. S. Taur and S. Y. Kung, "Fuzzy-decision neural networks," In *Proc. IEEE Conf. Acoustics, Speech, and Signal Process.* Minneapolis, Apr. 1993.

[21] ———, "Prediction-based networks with ecg application," In *Proc. IEEE Int. Conf. Neural Networks*, San Francisco, 1993.

[22] Li Wang and Dong-Chen He, "Texture classification using texture spectrum," *Pattern Recognition*, vol. 23, no. 8, pp. 905–910, 1990.

Sun-Yuan Kung (S'74–M'78–SM'84–F'88) received the Ph.D. degree in electrical engineering from Stanford University.

In 1974, he was an Associate Engineer of Amdahl Corporation, Sunnyvale, CA. From 1977 to 1986, he was a Professor of Electrical Engineering-Systems at the University of Southern California, L.A. In 1984, he was a Visiting Professor of Stanford University and Delft University of Technology. Since 1987 he has been a Professor of Department of Electrical Engineering at Princeton University. His research interests include spectrum estimations, digital signal/image processing, VLSI array processors, and neural networks. Since 1990 Dr. Kung served as an Editor in Chief of the *Journal of VLSI Signal Processing*. He is now an Associate Editor of the IEEE TRANSACTIONS ON NEURAL NETWORKS. He served as a member of IEEE Signal Processing Society's Administration Committee (1989–1991). He is presently serving on Technical Committees on VLSI Signal Processing and on Neural Networks. He has served as a founding member and General Chairman of various IEEE Conferences, including IEEE Workshops on VLSI Signal Processing in 1982 (L.A.) and in 1986 (L.A.), International Conference on Application Specific Array Processors in 1990 (Princeton) and 1991 (Barcelona), and IEEE Workshops on Neural Networks and Signal Processing in 1991 (Princeton) and 1992 (Copenhagen). He was the Keynote Speaker for the First International Conference on Systolic Arrays, Oxford, 1986, and International Symposium on Computer Architecture and DSP, Hong Kong, 1989. He has authored more than 250 technical papers and a textbook *"VLSI Array Processors"* (Prentice-Hall, 1988), and a second textbook *"Digital Neural Networks"* (Prentice-Hall, 1993). Dr. Kung was the recipient of the 1992 IEEE Signal Processing Society's Technical Achievement Award and was appointed as an IEEE-SP Distinguished Lecturer in 1994.



Jinshiuh Taur received the B.S. and M.S. degrees in electrical engineering from National Taiwan University, Taipei, Taiwan, R.O.C., in 1987 and 1989, respectively, and the Ph.D. degree in electrical engineering from Princeton University, in 1993.

In that same year, he was with Siemens Corporate Research, Inc., Princeton, NJ. He is currently an Associate Professor at the Department of Electrical Engineering, National Chung-Hsing University, Taiwan. His primary research interests include neural networks, pattern recognition, computer vision, and fuzzy logic systems.