Divide-and-Conquer Learning and Modular Perceptron Networks

Hsin-Chia Fu, *Member, IEEE,* Yen-Po Lee, Cheng-Chin Chiang, and Hsiao-Tien Pao

***Abstract*—A novel modular perceptron network (MPN) and divide-and-conquer learning (DCL) schemes for the design of modular neural networks are proposed. When a training process in a multilayer perceptron falls into a local minimum or stalls in a flat region, the proposed DCL scheme is applied to divide the current training data region (e.g., a hard to be learned training set) into two easier (hopely) to be learned regions. The learning process continues when a self-growing perceptron network and its initial weight estimation are constructed for one of the newly partitioned regions. Another partitioned region will resume the training process on the original perceptron network. Data region partitioning, weight estimating and learning are iteratively repeated until all the training data are completely learned by the MPN. We have evaluated and compared the proposed MPN with several representative neural networks on the two-spirals problem and real-world dataset. The MPN achieves better weight learning performance by requiring much less data presentations (99.01% $\sim$ 87.86% *lesser*) during the network training phases, and better generalization performance (4.0% *better*), and less processing time (2.0% $\sim$ 81.3% *lesser*) during the retrieving phase. On learning the real-world data, the MPN's show less overfitting compared to single MLP. In addition, due to its self-growing and fast local learning characteristics, the modular network (MPN) can easily adapt to on-line and/or incremental learning requirements for a rapid changing environment.**

***Index Terms*—Divide-and-conquer learning, modular perceptron network, multilayer perceptron, weight estimation.**

I. Introduction

MULTILAYER perceptrons (MLPs) have been widely used in various types of applications [21] due to their abilities to capture the underlying global structure and to generalize on untrained data. However, two issues are usually encountered in applying backpropagation learning on some fairly difficult problems: 1) how to successfully and fast converge to a learning goal and 2) how to predict a proper network size (e.g., the number of hidden neurons or layers). Alternative approaches have been proposed in pursuit of faster learning algorithms and efficient learning architectures. Hanson proposed the Meiosis networks [13] and Wynne-Jones [29] used the node splitting methodology to alternate hidden node sizes and structures to achieve a flexible learning architecture in an MLP. Cascade-correlation networks [9] were proposed to recursively augment hidden nodes and hidden layers of an MLP. According to these methods, an MLP can adjust its hidden nodes as well as hidden layer so that the networks can learn a fairly difficult problems, such cases include problems possessing strong nonlinearities, problems with many inputs or systems with many state variables, or problems having a large noise component. However, sometimes, these methods may expand the size of an MLP larger than necessary. Alternatively, committee machines were proposed. The idea of using a committee machine to realize a complex task may be traced back to Nilsson [24] in 1965; the network consisted of a layer of elementary perceptrons followed by a vote-taking perceptron in the second layer. This approach is based on commonly used engineering principle: divide and conquer. According to this principle, a complex supervised learning is achieved by distributing the learning task among a number of experts (a simple MLP), which in turn divides the input space into a set of subspaces. The combination of experts is said to constitute a committee machine. Committee machines can be classified into two major categories [15].

1) *Static Structure:* In this class of committee machines, the responses of several experts are combined by means of a method that does not involve the input signal. Major approaches in this class are:
   - *ensemble averaging* [23], [14], where the outputs of different experts are linearly combined to produce an overall output;
   - *boosting* [10], [6], [7], [28], [27], where a weak learning algorithm is converted into one that achieves arbitrarily high accuracy.

   The ensemble averaging or boosting based committee machines rely on the learning algorithm itself to do the integration. Ensemble averaging improves error performance by: 1) reducing of error due to bias by purposely overfitting the individual experts or 2) reducing of error due to variance by using different initial conditions in the training of the individual experts, and then ensemble-averaging their respective outputs. Boosting improves error performance: 1) by filtering [27], [10] the distribution of the input data in a manner causing the weak learning models (i.e., experts) to eventually learn the entire distribution or 2) by resampling the training examples according to a certain probability distribution as in the AdaBoost [11], [8]. The advantage of AdaBoost over boosting by filtering is that it works with a training sample of fixed size.

Manuscript received September 20, 1999; revised August 15, 2000. This work was supported in part by the National Science Council under Grant NSC 88-2213-E009-052.

H.-C. Fu and Y.-P. Lee are with the Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan 300, R.O.C.

C.-C. Chiang is with the Department of Computer Science and Information Engineering, National Dong Hua University, Hualien, Taiwan 974, R. O.C.

H.-T. Pao is with the Department of Management Science, National Chiao Tung University, Hsinchu, Taiwan 300, R.O.C.

Publisher Item Identifier S 1045-9227(01)02042-2.

2 *Dynamic Structure:* Instead of having a global network, several modular nets are used to learn the whole input space. Also, the input space is split up into several regions, and each modular net is trained to approximate the data in the designate region. The outputs of the modular nets are mediated by an integrating unit to produce the final output. Major architectures in this class are:
   - *mixture of experts* (ME) [16]: Mixture of experts claim that it can be viewed either as a modular version of multilayer supervise network, and/or as an associative version of competitive learning. In this design, the training of a local expert on a given training case can alleviate interference from the weights within other local expert;
   - *gated expert* [1]: Instead of using a probabilistic formulation in the gating network [16], an algorithm called optimization theory framework was adopted in the gated network to combine all the outputs of the expert networks to produce the final output. The algorithm also slices up the input space and approximates each of the partitioned regions using an expert network, separately. When comparing with the backpropagation algorithm, this method achieved considerable improvements particularly for hard problems;
   - *hierarchical mixture of experts* (HME) [17]: In this architecture, the individual outputs of the experts are combined by means of several gating networks arranged in a hierarchical manner.

As to previous research on modular neural networks, we recommend two papers [2], [25] and references therein for a more complete survey. The advantages of using various types of mixture of experts are: 1) splitting the whole input space into smaller regions can accelerate the learning processes; 2) partitioning a global network into several modular can imply a reduction of the number of parameters (i.e., weights); 3) using one module to learn one subregion of input data can avoid the interference matters, such as temporal and/or spatial crosstalk.

However, some problems remain to be solved so that modular network can be efficiently implemented for various applications:

1) how to split the input space properly, such that the decomposition can be beneficial to both learning and generalization;
2) how to decide the proper number of experts in a committee machine for a particular task.

Frequently, these problems were handled by trial and error. In this paper, we propose a new approach for the design of an MLP based modular neural networks, the modular perceptron network (MPN).

In this new approach, we have designed two learning algorithms: one is the error correlation based *divide-and-conquer learning* (DCL) scheme [3], and the other is the *weight estimation* (WE) method [20]. According to the error correlation scheme, the DCL divides a complex training data set into two subsets, one is an easy to learn region, and the other is a hard to learn region. And, then a new MLP is created to learn the hard region, in the meantime the original MLP continues to learn the easy region. The divide and conquer process continues until all the training data are learned successfully. It can be seen that the number of input data subregions and the number of MLP subnets were created by the system itself, instead of being predicted by neural network designers or users. Before the standard error backpropagation learning process is conducted in a subnet, weight estimation for the subnet is applied first. The WE method, which is motivated by Oriented principal component analysis [5], seeks to guide the initial weight vector toward the desired orientation, such that faster weight learning and less subnet creating can be achieved during the construction of the MPNs.

To observe the feasibility and effectiveness of the proposed modular network, we applied DCL to learn the two-spirals problem (TSP) [19] and to classify the Pima Indians diabetes dataset [22] for purposes of evaluation and comparison. According to our experimental results, DCL with the WE scheme can effectively deal with the slow learning and the unpredictable network size (i.e., the number of hidden units) problems in the design of an MLP-based system. Compared to the cascade correlation network [9] with 17 (or 10) layers and a conventional 2–30–1 neural network using hybrid fast learning schemes [18], the learning time of an MPN is about 8.24 to 108.14 times faster. In order to see the quality of the decision boundary generated by the proposed methods, we present here a few sets of testing data, which are located around the original TSP sample data (cf. Section III-A-2). The average generalization performance of the MPN can reach 97.97%. However, the average generalization performance of the cascade-correlation networks can only reach 96.25%. On the Pima dataset, the MPN achieves an averaging training and testing performance at 78% and 76%, respectively. This result seems to outperform Adaboost slightly [11] but this comparison is less compelling.

The rest of this paper is organized as follows. Section II presents the architecture of the MPN, DCL and the WE method. In Section III, simulation results are presented and compared with some other neural networks. Finally, concluding remarks are given in Section IV.

II. Modular Perceptron Networks

Most of previously proposed modular neural networks (MNNs) demand designers and users to specify the number of subnets during system configuration. However, because a user may not know the target problems very well, then this specification would be difficult for estimate. Unlike existing MNN, the proposed MPN does not require the designer or the user to specify the number of subnets for solving any given problems. The designer just needs to specify a general architecture for the MPN (cf. Section II-A). The proposed DCL scheme can automatically create new subnets according to the particular problem's needs during the learning phase. The design of the DCL scheme and MPN is also motivated by the concept of divide-and-conquer strategy. Initially, the MPN contains only one subnet. When the learning status of the subnet cannot be improved further, the MPN partitions the data set into two subsets, and then automatically adds a new subnet into the

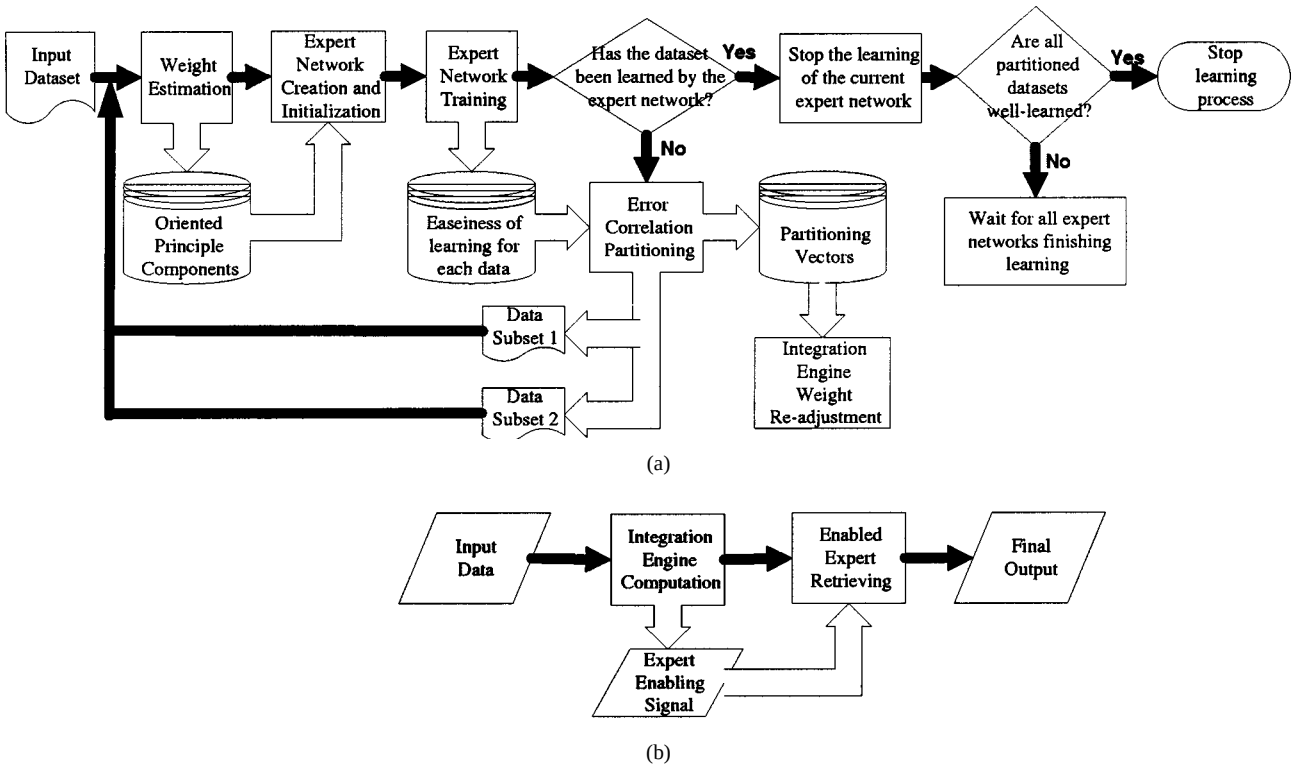

Fig. 1. DCL algorithm processes flow diagram. (a) Learning phase of the DCL modular perceptron network. (b) Retrieving phase of the DCL modular perceptron network.

MPN. After the data partitioning, the MPN has two subnets for the learning of the two smaller data subsets. The partitioning process continues until the whole training data are well learned. According to the described learning scheme, three important issues have to be addressed during the learning and retrieving phases of the network. For the learning phase, the first issue is how to partition the data set in a systematic and reasonable way. Blind partitioning may not be of any help to the learning of the target problem. The second issue is how to initialize the newly added subnets efficiently. Random initialization might trap the subnets into a local optimum very frequently and thus results in many unnecessary partitioning and redundant subnets. For the retrieving phase, the most important issue is how to derive the final output from these subnets. Improper integrating scheme would degrade the retrieving speed and accuracy. Based on the addressed issues, we proposed the following schemes for the proposed DCL and the MPN. The first is the error correlation partitioning (ECP) which is used for partitioning the input dataset. Instead of partitioning the data according to the *label* of each data, the ECP partitions the data according to the *ease of learning* of each dataset. In other words, the purpose of ECP is to partition the training data set into two groups, the easy-to-learn data group and the difficult-to-learn group. The ideal result of the ECP would be that only the difficult-to-learn data subset needs to be further trained or partitioned, while the easy-to-learn data subset would be well-learned after the partitioning. The second scheme proposed for the DCL modular perceptron network is the initial WE method for the of newly created subnets. The design of WE scheme is motivated by the observation that a weight vector of an MLP classifier acts like a partition plane, separating different classes of a data set. In Section II-C, we propose a method to find the partition plane and to convert its parameters into the initial weight vector. By this way, the learning process in each subnet can be faster and, most importantly, redundant subnet creation can be significantly reduced. The third scheme is the integration method of all subnets' outputs. The DCL modular perceptron network uses an integration engine (IE) to derive the final output from the multiple cooperative subnets. The basic function performed in IE is to identify the location in the problem space for the input data and then select the appropriate subnet to generate the output. Since the IE needs to associate the input data with the corresponding subnet, IE must be designed tightly coupled with the ECP. Fig. 1 depicts a high-level flow diagram for the learning and retrieval phases of the DCL modular perceptron network. The details of the operations are provided in subsequent sections.

A. Network Architecture

The architecture of the proposed MPN is shown in Fig. 2. As mentioned, the MPN consists of two modules: the DCL engine and the IE. The DCL engine, which consists of a set of self-growing MLP subnets, performs training data partitioning, subnet self-growing and weight learning. The IE is a self-growing two-layer feed-forward neural network with the *Heaviside* (hard-limit) activation function at each hidden and output neurons. Acting as a gating network of the MPN, the IE performs the function of a mediator among the subnets in the DCL engine. The number of output neurons of the IE is equal to the number of subnets in the DCL engine. During the learning phase, each subnet in the DCL engine learns its designated subset of training data, respectively. When the learning process

of a subnet falls into a local minimum, the current training data set is then partitioned according to the proposed *ECP* scheme (cf. Section II-B1) into two subsets. In the meantime, at the IE side, a new output neuron and a hidden neuron are also created to provide the mediating function for the newly created subnet in the MPN. During the retrieving phase, when the IE receives an input pattern, it generates proper control signals to select a proper subnet in the DCL engine for the current input pattern.

The system dynamics of the MPN during the retrieving phase can be formulated as follows:

$$h_i(\boldsymbol{x}) = 2\mathcal{H}(w_{i,0} + \boldsymbol{w_i} \cdot \boldsymbol{x}) - 1 = \begin{cases} -1 & \text{if } w_{i,0} + \boldsymbol{w_i} \cdot \boldsymbol{x} < 0 \\ 1 & \text{if } w_{i,0} + \boldsymbol{w_i} \cdot \boldsymbol{x} \geq 0, \end{cases} \tag{1}$$

$$c_i(\boldsymbol{x}) = \mathcal{H}\left(u_{i,0} + \sum_{j=1}^{N-1} u_{i,j} h_j(\boldsymbol{x})\right) = \begin{cases} 0 & \text{if } u_{i,0} + \sum_{j=1}^{N-1} u_{i,j} h_j(\boldsymbol{x}) < 0 \\ 1 & \text{if } u_{i,0} + \sum_{j=1}^{N-1} u_{i,j} h_j(\boldsymbol{x}) \geq 0 \end{cases} \tag{2}$$

$$o(\boldsymbol{x}) = \sum_{i=1}^{N} c_i(\boldsymbol{x}) F_i(\boldsymbol{x}) \tag{3}$$

where $\mathcal{H}(\,)$ denotes the *Heaviside* function. As shown in Fig. 2, let $\boldsymbol{x}(\in \Re^n)$ and $\boldsymbol{o}$ represent the input and the output vectors of an MPN. Assume that a trained MPN contains N subnets in the DCL engine; then, there are N output and $N-1$ hidden neurons in the IE. $F_i(\boldsymbol{x}) = (F_{i,1}(\boldsymbol{x}), \ldots, F_{i,L}(\boldsymbol{x}))$ denote the output vectors of the ith subnet of the DCL engine, where L is the dimension of the output vector. Let $c_i(\boldsymbol{x})$ denote the control signal generated by the ith output neuron of the IE, and let $h_i(\boldsymbol{x})$ denote the output of the ith hidden neuron of the IE. For the connection weights in the IE, $\boldsymbol{w_i}(=w_{i,1}, \ldots, w_{i,n})$ and $\boldsymbol{u_j}(= u_{j,1}, \ldots, u_{j,N-1})$ denote the weight vector of the ith hidden neuron and the weight vector of the jth output neuron, respectively. In addition, $w_{i,0}$ and $u_{j,0}$ represent the biases of the ith hidden neuron and jth output neuron, respectively.

B. Learning Process of the Modular Perceptron Network

The learning process of the proposed MPN consists of two concurrent learning processes: the learning of the DCL Engine and the learning of the IE. We will describe the data partition scheme first, and then the training scheme of this two engines in the following sections.

1) DCL and the Error Correlation Partition: In the DCL engine, each subnet, concurrently learns its corresponding part of the training data during the learning phase. The learning phase for each subnet can adopt almost any type of supervised learning scheme (e.g., LMS, BP, ...). When a subnet successfully learns its designated training data subset, i.e., the error (MSE) of each training pattern is less than a prescribed error tolerance, then the subnet stops its learning process. On the other hand, if a subnet can not continuously reduce its training error at a predetermined rate, then the current training data set is suggested

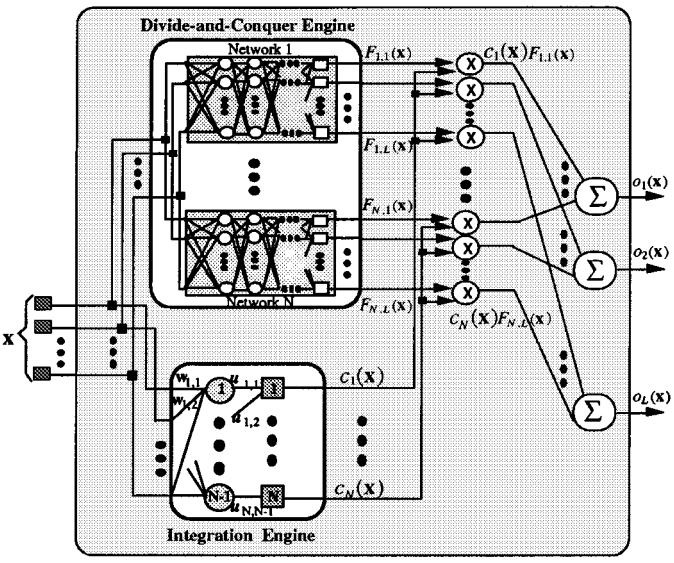

Fig. 2. The architecture of an MPN: (a) DCL engine, (b) IE, and (c) the output control logics [to the right of (a) and (b)].

to be partitioned into two subsets. Frequently, if the partitioning of the training data is carried out without referencing the previous training status; then, the partitioned data subsets may still be a hard-to-learn data region. In this situation, the following learning process may be even more difficult and yet still slow after data region partitioning.

In order to have efficient partitioning for a training data set, analyzing the training status of the current subnet and its corresponding training data set before the partitioning process is necessary. Heuristically, the training error of each training pattern is a good quantitative indicator of the "easiness of learning" (EOL) for each individual training pattern. Basically, a training set can be partitioned into two subsets, one of which contains patterns with good EOL (small training error) while the other subset contains patterns with poor training qualities. For the partitioned subset with good EOL, the current subnet should be a desired candidate for this part of the training data. If the training quality of the subset is not good enough, a new subnet with a proper estimated initial weight vector seems a good choice for the proceeding training process. Based on this concept, we propose the error correlation partitioning (ECP) scheme to EOL. In the ECP, a cost function E_c is defined as

$$E_c = \left(\sum_{i=1}^{P} \left(E_i - \overline{E}\right) \left[\mathbf{w}^T \left(\boldsymbol{x_i} - \overline{\boldsymbol{x}}\right)\right]\right)^2 \tag{4}$$

where

- P number of training data in the current subset to be partitioned;
- $\mathbf{w}$ normalized linear projection vector, i.e., $\|\mathbf{w}\| = 1$;
- $\overline{\boldsymbol{x}}$ mean of the current training data, i.e., $\overline{\boldsymbol{x}} = 1/P \sum_{i=1}^{P} \boldsymbol{x}_i$.

Let E_i denote the training error of the training pattern $\boldsymbol{x_i}$, and $\overline{E}$ denote the average error of the training subset. The basic goal of ECP is to find a linear projection vector $\mathbf{w}^*$ such that E_c is maximized. As shown in (4), in order to maximize E_c, both terms $(E_i - \overline{E})$ and $[\mathbf{w}^{*T}(\boldsymbol{x_i} - \overline{\boldsymbol{x}})]$ need to have the same sign (i.e., both

in positive or in negative values). In other words, if we project the vector $(\mathbf{x}_i - \overline{\boldsymbol{x}})$ along the unit vector $\mathbf{w}^*$, i.e., $[\mathbf{w}^{*T}(\boldsymbol{x_i} - \overline{\boldsymbol{x}})]$, then the projection vector of patterns with training error less than the average error $\overline{E}$ should be in the opposite direction with respect to the projection direction of the patterns with training error larger than the average error $\overline{E}$. Moreover, along with the proposed projection direction, most of the patterns in the same category will have the same sign of projection value. Fig. 3 depicts this concept of partition scheme, where the training data is partitioned into two categories S_0 and S_1 with minimum correlation error. The ECP scheme can be formally described using the following rules:

R1: if $\mathbf{w}^{*T}\boldsymbol{x_i} < \mathbf{w}^{*T}\overline{\boldsymbol{x}}$, then $\boldsymbol{x_i} \in S_0$;
R2: if $\mathbf{w}^{*T}\boldsymbol{x_i} \geq \mathbf{w}^{*T}\overline{\boldsymbol{x}}$, then $\boldsymbol{x_i} \in S_1$.

In the following, we will proceed to find the linear projection vector $\mathbf{w}^*$. The error correlation cost function E_c (4) can be further derived as follows:

$$\begin{aligned} E_c &= \left(\sum_{i=1}^{P}(E_i - \overline{E})\left[\boldsymbol{w}^T(\boldsymbol{x_i} - \overline{\boldsymbol{x}})\right]\right)^2 \\ &= \left(\sum_{i=1}^{P}(E_i - \overline{E})\left[\boldsymbol{w}^T(\boldsymbol{x_i} - \overline{\boldsymbol{x}})\right]\right) \\ &\quad \cdot \left(\sum_{i=1}^{P}(E_i - \overline{E})\left[\mathbf{w}^T(\boldsymbol{x_i} - \overline{\boldsymbol{x}})\right]\right) \\ &= \mathbf{w}^T\left[\sum_{i=1}^{P}(E_i - \overline{E})(\boldsymbol{x_i} - \overline{\boldsymbol{x}})\right] \\ &\quad \cdot \left[\sum_{i=1}^{P}(E_i - \overline{E})(\boldsymbol{x_i} - \overline{\boldsymbol{x}})^T\right]\mathbf{w}. \end{aligned}$$

Let $\mathbf{S} = [\sum_{i=1}^{P}(E_i - \overline{E})(\boldsymbol{x_i} - \overline{\boldsymbol{x}})] \cdot [\sum_{i=1}^{P}(E_i - \overline{E})(\boldsymbol{x_i} - \overline{\boldsymbol{x}})^T]$; then

$$E_c = \mathbf{w}^T\mathbf{S}\mathbf{w}. \tag{5}$$

Since $\|\mathbf{w}\| = 1$, the weight vector $\mathbf{w}^*$ which maximizes the error correlation E_c is equal to the first eigenvector (corresponding to the largest eigenvalue) of the error correlation matrix $\mathbf{S}$.

2) Learning Scheme in the Integration Engine: As indicated in (2), the function of an IE is to generate the control signal $c_i(\boldsymbol{x})$ to enable the right subnet to derive the final output, given an input $\boldsymbol{x}$. Thus, the major learning task of the IE is to obtain appropriate values for its two types of weights, i.e., $\boldsymbol{w_i}$s and $\boldsymbol{u_i}$s. Corresponding to the ECP processes during the learning phase of the DCL engine, the IE synchronously generates a hidden neuron and an output neuron for each partitioning occurred in the DCL engine. The new hidden neuron actually acts as a partition hyperplane in the input space. Consequently, the weight vector $(\boldsymbol{w_i})$ and the bias $(w_{i,0})$ of the hidden neuron i in the IE should be determined according to the projection vector $\mathbf{w}^*$ derived from the ECP rules R1 and R2, i.e.,

$$\boldsymbol{w_i} = \mathbf{w}^*, \quad \text{and} \quad w_{i,0} = -\mathbf{w}_i^{*T}\overline{\boldsymbol{x}}.$$

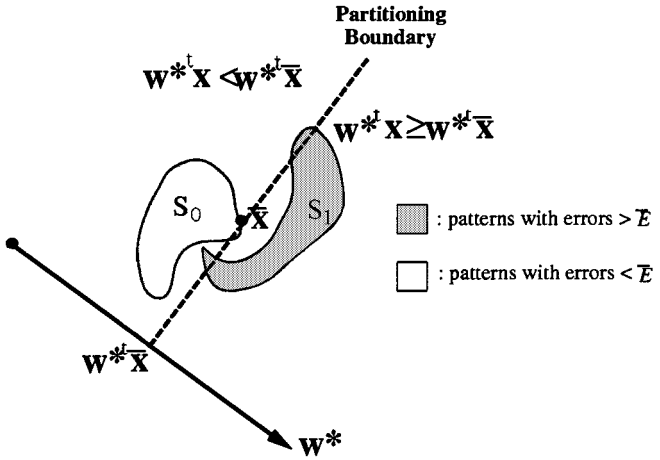

Fig. 3. Graphical representation of the ECP of two data sets S_0 and S_1.

After determining the weight vector of the new hidden neuron, it is not difficult to derive the connection weights between the new hidden neuron and the output neurons based on the concepts of decision tree. We leave the detail derivation in the Appendix. To summarize the whole algorithmic steps of the IE learning, the algorithm is listed as follows.

Algorithm 1: Learning Algorithm for the Integration Engine

1: Initialize the first output neuron c_1 with a fixed value 1; connect the output neuron directly to the input layer for the moment (i.e., no hidden neuron yet).
2: While ECP is performed in the DCL engine, execute Step 3.
3: Assume that the training data subset S_i with an identification string I_i $(= s_{i1}s_{i2}\ldots s_{id_i})$ is to be partitioned. Let $\mathbf{w}_i^*$ be the projection vector obtained by the ECP method, and let $\overline{\mathbf{x}}_i$ be the mean vector of S_i. Insert a hidden neuron, say h_k, at the hidden layer of the IE, and set its bias $w_{k,0}$ and weight vector $\boldsymbol{w_k}$ as

$$\boldsymbol{w_k} = \mathbf{w}_{\boldsymbol{i}}^*, \quad \text{and} \quad w_{k,0} = -\mathbf{w}_{\boldsymbol{i}}^{*t}\overline{\mathbf{x}}_{\boldsymbol{i}}.$$

Proceed to Steps 3a∼3b. (Assign weights between output and hidden neurons.)
3a: If there is only one output neuron c_1 in the output layer, then perform Steps 3a-(i)∼3a-(v); otherwise, perform Step 3b.
3a-(i): Create a new output neuron c_2.
3a-(ii): Connect c_1 to the hidden neuron h_1 with a connection weight -1 (i.e., $u_{1,1} = -1$) and set the bias of c_1 as -1 (i.e., $u_{1,0} = -1$).
3a-(iii): Connect c_2 to the hidden neuron h_1 with a connection weight one

(i.e., $u_{2,1} = 1$) and set the bias of c_2 as -1 (i.e., $u_{2,0} = -1$).

3a-(iv): Set the identification string of the new subset of training data corresponding to c_1 as $\overline{h}_1$ and set the path depth d_i of this new subset as one.

3a-(v): Set the identification string of the new subset of training data corresponding to c_2 as h_1 and set the path depth d_i of this new subset as one.

3b: Let c_i be the output neuron corresponding to the original unpartitioned subset S_i. Perform Steps 3b-(i)$\sim$3b-(vi).

3b-(i): Create a new output neuron c_j.

3b-(ii): Assign the weight vector $\boldsymbol{w_i}$ to the weight vector of h_j.

3b-(iii): Set the identification string of the new subset of training data corresponding to c_i as $s_{i1}s_{i2}\ldots s_{id_i}\overline{h}_k$ and increase the depth d_i of this new subset by one.

3b-(iv): Set the identification string of the new subset of training data corresponding to c_j as $s_{i1}s_{i2}\ldots s_{id_i}h_k$ and increase the depth d_i of this new subset by one.

3b-(v): Connect c_i to the new hidden neuron h_k with connection weight -1 (i.e., $u_{i,k} = -1$) and set the bias of c_i as $-d_i$, (i.e., $u_{i,0} = -d_i$).

3b-(vi): Connect c_j to the new hidden neuron h_k with connection weight one (i.e., $u_{j,k} = 1$) and set the bias of c_j as $-d_i$, (i.e., $u_{j,0} = -d_i$).

C. Weight Estimation for DCL

This section introduces the weight estimation method that can reduce the number of data presentation and the number of subnets during the DCL process in an MPN.

Basically, a weight vector of an MLP classifier acts like a data partition hyperplane separating different classes of data in the input space. As shown in Fig. 4, a weight vector represented by plane A can be one of the partition planes for the two classes ($\times$, $\circ$). Suppose plane B corresponds to a randomly initialized weight vector. Since plane B is nearly perpendicular to plane A, then training the weight vector representing plane B to a weight vector representing plane A may require a lot of training iteration. It is clear to see that if one can initialize a weight vector at or close to plane A, then training process can be much shorter. However, for real-world applications, the data class distribution can be highly nonlinear. To construct a partition plane like the plane A in Fig. 4, is not straightforward. In this section, we propose a method, which is motivated from principal components analysis (PCA) to roughly partition the data space into two classes. And then, its parameters are used to initialize the weight vector of an MLP. In addition, during DCL process, the data space is repeatly partitioned according to the ECP scheme.

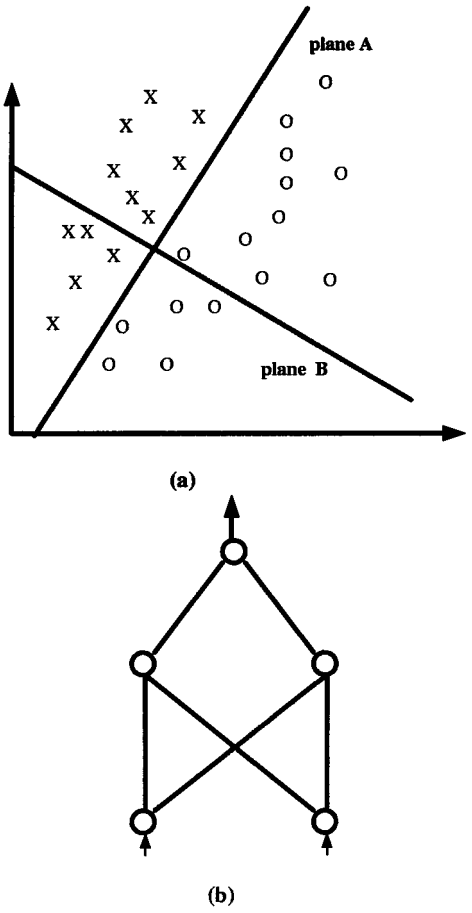

Fig. 4. (a) Class distribution of two data classes $\times$ and $\circ$. Plane A corresponds to a well-trained weight vector of the MLP in (b), and plane B represents a randomly initialized weight vector. (b) An MPN contains one hidden layer and two hidden nodes.

Thus, a highly nonlinear data space, e.g., the two classes in TSP [19], will turn into less nonlinearly or even linearly separable after a few partition cycles. Therefore, the proposed weight estimation method can significantly improve the learning performance of the DCL MPNs.

Without loss of generality, we use an MLP containing one hidden layer with two hidden nodes as the basic MLP based subnet for the MPN. In [5], Diamantaras and Kung proposed oriented principal components analysis (OPCA), which can optimally separate two data classes along the direction of the oriented principal component. In this paper, we apply the OPCA method to initiate the orientation of the weight vector. Supposedly, the data classes $\boldsymbol{z}$ and $\boldsymbol{v}$ are used to train the weight vector of an MLP. We first find the direction vector $\boldsymbol{w}$ that maximizes the energy ratio of $\boldsymbol{z}$ and $\boldsymbol{v}$

$$J_{opc} = \frac{E\{(\boldsymbol{w^T z})^2\}}{E\{(\boldsymbol{w^T v})^2\}} = \frac{\boldsymbol{w^T R_z w}}{\boldsymbol{w^T R_v w}} \tag{6}$$

where $\boldsymbol{R_z} = \{\boldsymbol{zz^T}\}$ and $\boldsymbol{R_v} = \{\boldsymbol{vv^T}\}$. The solution $\boldsymbol{w}^*$ to (6) is called the principal oriented component of the two classes, which means that when the principal component of $\boldsymbol{z}$ is steered by the distribution of $\boldsymbol{v}$, it will be oriented toward the directions

where $\boldsymbol{v}$ has minimum energy while an attempt is made to maximize the projection energy of $\boldsymbol{z}$. The oriented principal component is the principal generalized eigenvector of the symmetric generalized eigenvalue problem

$$\boldsymbol{R_z w} = \boldsymbol{\lambda R_v w}. \tag{7}$$

Where $\boldsymbol{w} = \{w_1, w_2, \ldots, w_n\}$ and $\boldsymbol{\lambda} = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$. In addition, all the components w_i that maximize (6) should satisfy the following constraints:

$$w_i{}^T R_v w_j = 0,\ w_i{}^T R_z w_j = 0, \qquad \forall\, j < i. \tag{8}$$

For initialization of the weight vector of an MLP based subnet, the largest generalized eigenvector w_i is assigned to all the hidden nodes in the MLP.

Through oriented principal component analysis, we can achieve only the direction of the hyperplane for separating the data space into two classes. However, the proper location for the hyperplane along the OPC is still to be decided. According to the Bayesian decision rule, if the distribution of data sets $\boldsymbol{z}$ and $\boldsymbol{v}$ on the OPC axis is known, then the point where two probability density curves cross will be the solution of discrimination for these two data sets. As shown in Fig. 5, if the OPC axis of the two sets is determined, then the distribution of these two data sets can be also estimated by projecting on the axis. Suppose the minimum error position for classification is b_k; the decision boundary can be constructed as follows.

Let $\boldsymbol{z'}$ and $\boldsymbol{v'}$ be the projection of $\boldsymbol{z}$ and $\boldsymbol{v}$ on the OPC axis, i.e., $\boldsymbol{z'} = \boldsymbol{w^T z}$, and $\boldsymbol{v'} = \boldsymbol{w^T v}$.

Assume the projected data have a D-dimension Gaussian distribution, with means and variances as $\boldsymbol{\mu_{z'}}$, $\boldsymbol{\mu_{v'}}$, $\boldsymbol{\Sigma^2_{z'}}$, $\boldsymbol{\Sigma^2_{v'}}$, respectively. The conditional probability functions $p(z'|C_z)$, $p(v'|C_v)$ for z' and v' are

$$P(\boldsymbol{x}|C_y) = \frac{1}{(2\pi)^{D/2}\,|\Sigma_y|^{1/2}} \cdot \exp\left[-\tfrac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu_y})\,\overline{\sum_{\boldsymbol{y}}}^{\boldsymbol{1}}(\boldsymbol{x}-\boldsymbol{\mu_y})\right] \tag{9}$$

where

$\boldsymbol{x} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x_D}]$ input data;

$\boldsymbol{\mu_y} = [\boldsymbol{\mu_{y_1}}, \boldsymbol{\mu_{y_2}}, \ldots, \boldsymbol{\mu_{y_D}}]$ mean vector; and diagonal matrix

$\boldsymbol{\Sigma_y} = \text{diag}[\boldsymbol{\sigma^2_{y_1}}, \boldsymbol{\sigma^2_{y_2}}, \ldots, \boldsymbol{\sigma^2_{y_D}}]$ covariance matrix.

The index $\boldsymbol{y}$ can be either $\boldsymbol{z}$ or $\boldsymbol{v}$. P_z and P_v denote the prior probabilities of $\boldsymbol{z}$ or $\boldsymbol{v}$. By definition, $P_z + P_v = 1$.

Let us define a discriminate function ϕ as

$$\phi(\boldsymbol{x}|C_y) = \tfrac{1}{2}\sum_{d=1}^{D}[\log(1/\sigma^2_{y_d}) - (1/\sigma^2_{y_d})(x-\mu_{y_d})^2] - \frac{D}{2}\log 2\pi + \log P_y, \tag{10}$$

where $\boldsymbol{y}$ can be either $\boldsymbol{v}$ or $\boldsymbol{z}$.

The decision boundary $\boldsymbol{b_k}$ along OPC for classes z and v can be achieved by solving the following equation:

$$\phi(b_k|C_z) - \phi(b_k|C_v) = 0. \tag{11}$$

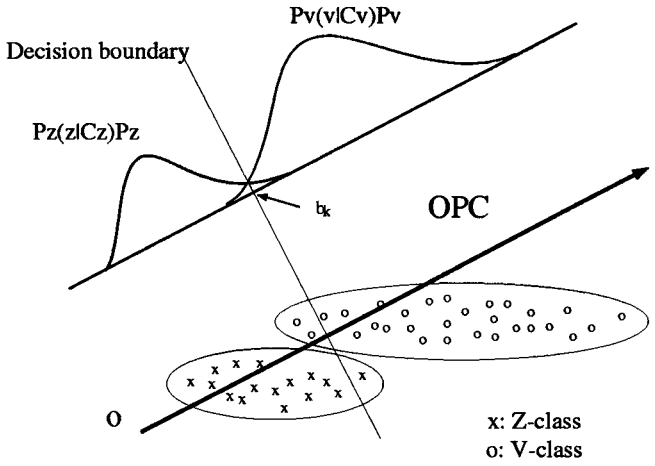

Fig. 5. Data projection and decision boundary of two data sets z and $\boldsymbol{v}$ along the OPC axis.

Thus, the initial weight and bias for the partition hyperplane $\boldsymbol{w^T x + b_k} = 0$ can be achieved.

As shown in Fig. 5, the partition hyperplane $\boldsymbol{w^T x + b_k} = 0$ decided by OPCA and the Bayesian decision rule seems to be a good initial partition hyperplane for the two data sets $\boldsymbol{z}$ and $\boldsymbol{v}$.

D. On-Line Learning of MPN

In this section, we will discuss that the MPN with WE is suitable for on-line and/or incremental learning problems. Sometimes, a well-trained neural network may occasionally receive new training samples in order to further improve its generalization capabilities. If the neural network learns only these new patterns, its generalization performance will be usually degraded seriously. To tackle this problem, neural networks have to keep all the original patterns as well as the new patterns. Obviously, this is not a desirable way to conduct the learning process because of the high computational complexity. In our proposed method, it is easy to see that when a new training pattern is added to the original training set, there is only one subnet that needs to be retrained. In other words, suppose a new pattern is in one of the partitioned subsets of the training set; then, the IE will select its corresponding subnet in the DCL Engine in order to perform the retraining process after weight estimation, and the rest of the subnets need not be retrained. If the allocated subnet can not learn the new training pattern in a few learning cycles, then an ECP partition process and a new subnet are created to learn the newcomer. In this situation, a simple subnet can learn much faster than a complex MLP when a new training pattern is coming. Thus, we can claim that the DCL of an MPN can model and learn a dynamically changing environment.

III. Comparative Simulations

In order to evaluate the learning and retrieving performance of the proposed MPN and DCL, two experimental results will be discussed. In the first part, we use the TSP [19] as the benchmark. The TSP is an extremely difficult problem for multilayer perceptron networks. As shown in Fig. 6, the training set consists of 194 X–Y values, which are arranged in two interlocking spirals that go around the origin for three times. The training

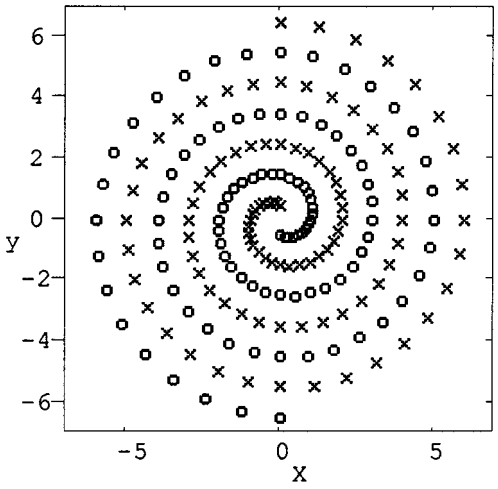

Fig. 6. The data distribution for the TSP with 194 training points.

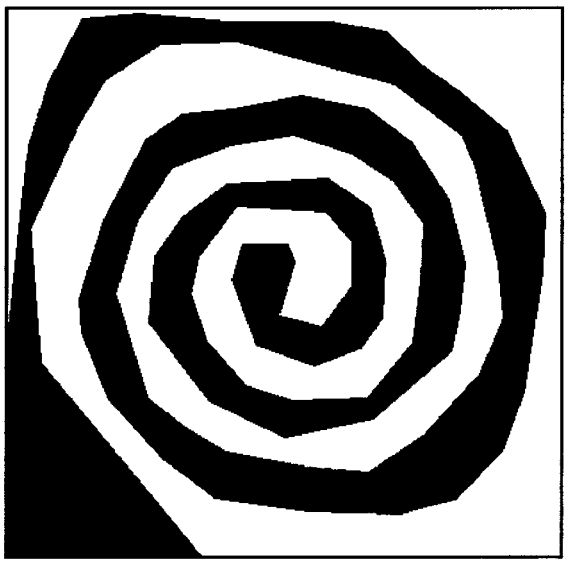

Fig. 7. The output classification image created by an MPN with 25 subnets and 40 012 data presentations.

goal is to develop a feedforward network with sigmoidal units that properly classifies these 194 training points in two classes. The second experiment explores the classification ability of the MPN on a real-world dataset, the Pima Indians diabetes dataset [22], which is available at the university of California-Irvine repository. We will show the generalization performance of the MPN before and after the training converges. This is an important issue, especially in applying flexible machine learning techniques to noisy real-world data.

A. Experiments on Two-Spirals Problem

1) Training Phase of MPN: In this two-spirals benchmark comparative study, we used two different MLPs as the kernel model of the MPN. One was a conventional MLP that contained one hidden layer with two hidden nodes, and the other was a modified MLP called the dynamic-threshold quadratic sigmoidal neural network (QSNN) [4]. QSNN had the same architecture as the conventional MLP except that it used a different activation function called the quadratic sigmoid function (QSF) [4] in each hidden neuron.

During the learning phase of the DCL engine, if the mean squared learning error could not be improved by 0.001 within 15 epochs, then the current training subset was partitioned according to the error correlation partitioning scheme. We also assigned the squared error tolerance ϵ to be 0.1225 (i.e., 0.35^2) for each training pattern. As long as the squared training error of a training pattern was larger than ϵ, this pattern was not considered to be correctly learned yet. Fig. 7 depicts the learning results obtained by an MPN for the TSP.

Experimental Results: The learning performance of several different types of neural networks are compared and listed in Table I. Each performance results were repeated for ten trials with random initial weight values. In a training process, one epoch is defined as a full presentation of each of the I/O patterns in the training set. It should be noted that it is appropriate to compare performance in terms of the number of presentations in the MPN. This is because the number of pattern presentations in each epoch will be gradually reduced as the DCL iterations continue, so the number of presentations is a true measure for problems that do not cycle through the entire training set between each weight-update iteration.

The cascade-correlation learning algorithm reported in [9] is able to produce reasonably good performance on this "two-spirals" benchmark. The cascade-correlation algorithm uses a sigmoidal activation function for both the output and hidden units and a pool of eight candidate units. All the trails were successful, requiring 1700 epochs or $1700 * 194 = 329\,800$ pattern presentations for 194 patterns. After the training phase, the number of hidden units built into the net varied from 12 to 19, with an average of 15.2 and a median of 15, which accounts for 168 connection weights. Fig. 8 shows the output of a 12-hidden-unit network based on cascade-correlation, as the input was scanned over the X–Y field. Karayiannis [18] proposed a hybrid learning scheme, which combines bottom-up unsupervised learning and top-down supervised learning techniques to achieve fast and efficient training of an MLP. By using the hybrid learning scheme on a 2–30–1 MLP, a TSP of 150 data points could be learned in 28 874 epochs, or $28\,847 * 150 = 4\,327\,050$ pattern presentations with a total error of 0.01. The proposed DCL technique with WE requires 21.1 ± 2.3 subnets, and 2082 ± 478 epochs or $40\,012 \pm 6905$ pattern presentations to learn Two-spirals problem. In other words, an MPN can learn the Two-spirals problem for much less data presentation (99.07% $\sim$ 87.86% lesser).

In the lower part of Table I, the learning performance of MPNs with or without weight estimation is presented. Applying the weight estimation method to the DCL for the MLP-based MPN, the number of the epochs presented and the number of subnets needed can be reduced by about 33.61% and 33.14% respectively, and the number of data presentations can be reduced 4.15%. As for the QSNN-based MPN, weight estimation can reduce the number of epoch and data presentation and subnet creation by 76.28%, 3.16%, and 28.23%, respectively. In general, the number of subnets required can be significantly reduced by applying weight estimation. Also, the weight estimation improves the learning performance of

TABLE I
EXPERIMENTAL RESULTS OF THE MLP WITH THE HYBRID LEARNING METHOD, CASCADE NEURAL NETWORKS AND VARIOUS MPN TYPES OF NEURAL NETWORKS USING DCL AND WE TO LEARN TWO-SPIRALS PROBLEM

Types of NNs	No. of epochs	No. of presentations	No. of subnets
Hybrid learning MLP	28847	4327050	NA
Cascade NNs	1700	329800	NA
MPN(MLP)	11266±1013	88525 ± 7703	52.8 ± 5.26
MPN(MLP)+WE	7494±744	84847 ± 4162	35.3 ± 3.6
MPN(QSNN)	2150±350	168714 ± 33960	29.4 ± 4.4
MPN(QSNN)+WE	2082±478	40012 ± 6905	21.1 ± 2.3

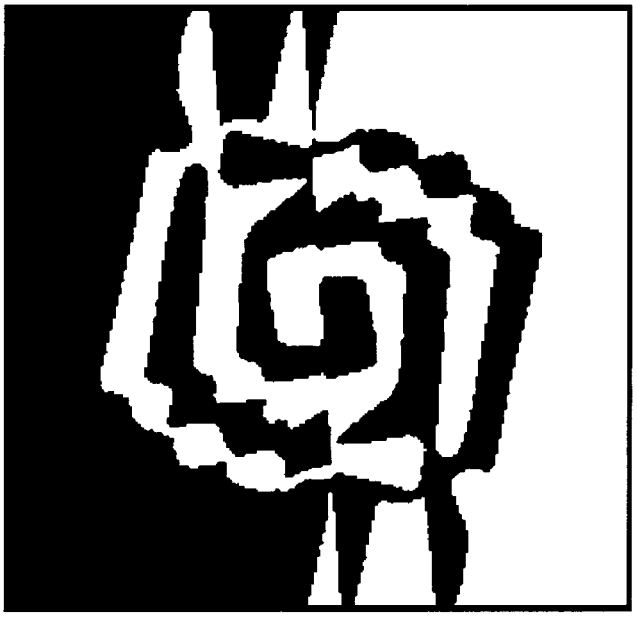

Fig. 8. The output classification image created by the cascade-correlation neural network [9] with 12 Sigmoid hidden units.

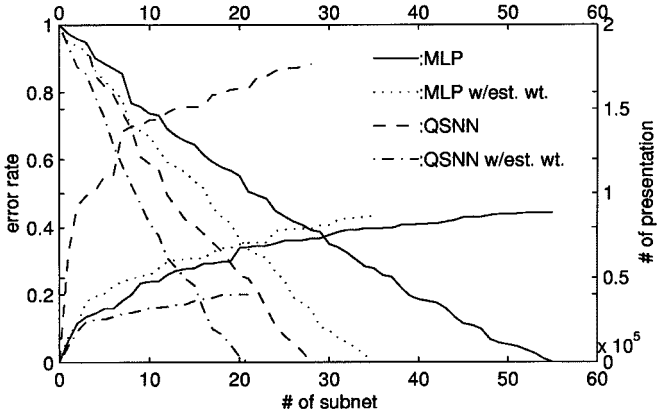

Fig. 9. The learning curves for the error rate and the number of data presentation with respect to the creation of subnets by four types of MPNs during learning for the TSP. The error rate is defined as the ratio between the number of unlearned patterns and total number of training patterns.

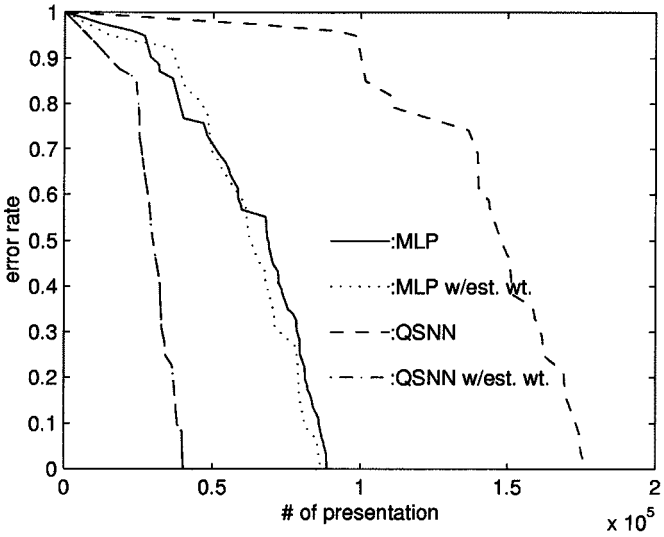

Fig. 10. The learning curves for the TSPamong four types of MPNs. The error rate is defined as the ratio between the number of unlearned patterns and total number of training patterns.

the MLP and QSNN-based MPNs significantly in terms of the number of presentations and the number of epochs required.

In the following, we would like to comment on the learning behavior of the MLP and QSNN-based MPNs in terms of DCL and weight estimation. As shown in Table I, an MLP-based MPN requires a lot of epochs to learn a TSP, however a QSNN requires much lesser epoch but more data presentations to achieve the same goal. In [4], it has been shown that a QSNN has much more learning and modeling power than an MLP. As shown in Fig. 9, at each creation of a subnet, the learning curve of the QSNN-based MPN shows lower error rate than an MLP-based MPN does. Basically, a QSNN-based MPN requires more data presentation to learn a given data subset before it claims learning failure and requests a new subnet creation. Hence, in terms of the number of subnet creation, a QSNN consumes more training presentation than an MLP-based MPN does.

As stated in Section II-C, the function of weight estimation is to direct the initial weight vector toward a desired orientation, such that faster weight learning can be expected. As shown in Figs. 9 and 10, by including the weight estimation in DCL, both the MLP and QSNN-based MPNs can achieve their learning goal for less number of epochs, data presentation and subnet creation. Especially, the weight estimation significantly reduces the number of data presentation for the QSNN-based MPNs.

2) Retrieving Phase of MPN: The generalization performance is an important index for evaluating the proposed MPN and the learning methods. We suggest the following procedures for generating testing data points for the TSP. Let l be the smallest distance between two sampling points that belong to two classes of TSP, and also multiply l by a few scale values (e.g., 0.1, 0.2, 0.3, 0.4) as the variance of a normal distribution centered at each training point of TSP. Fig. 11 depicts the

TABLE II
GENERALIZATION PERFORMANCE OF THE CASCADE NN AND VARIOUS TYPES OF MPNS FOR THE TWO-SPIRALS PROBLEM. "MLP-MPN" MEANS AN MLP BASED MPN. "WE" MEANS THE DCL ASSOCIATED WITH WE σS ARE THE VARIANCES OF NORMAL DISTRIBUTIONS CENTERED AT EACH SAMPLE POINT OF TSP. FOR EACH RANGE SPANNED BY DIFFERENT σ VALUES, 1940 TESTING DATA POINTS ARE GENERATED, AND THIS IS REPEATED FOR TEN TIMES TO EVALUATE THE GENERALIZATION PERFORMANCE

	Generalization Performance among four testing regions			
Type of NN	$\sigma = 0.1l$	$\sigma = 0.2l$	$\sigma = 0.3l$	$\sigma = 0.4l$
Cascade NN	99.20 ± 0.62	97.49 ± 1.10	95.38 ± 1.69	92.93 ± 2.18
MPN(MLP)	98.92 ± 0.37	97.80 ± 0.49	96.30 ± 0.61	94.36 ± 0.69
MPN(MLP)+WE.	98.92 ± 0.58	98.30 ± 0.50	97.35 ± 0.62	95.81 ± 0.59
MPN(QSNN)	99.31 ± 0.41	98.85 ± 0.51	97.54 ± 0.69	96.19 ± 0.88
MPN(QSNN)+WE.	99.41 ± 0.42	98.54 ± 0.70	97.41 ± 0.87	96.23 ± 0.68

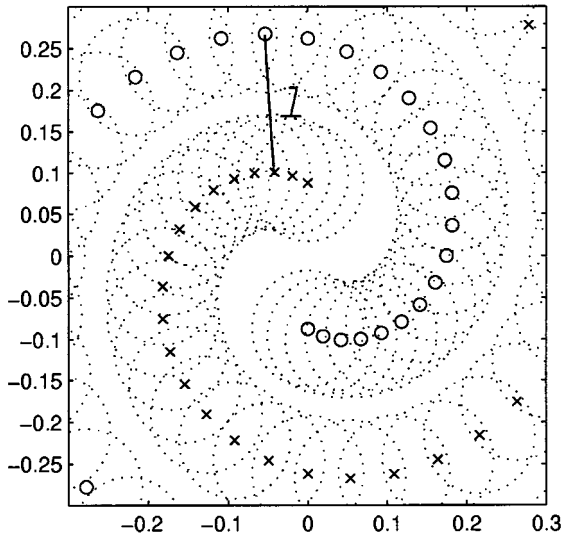

Fig. 11. Training and testing data points for generalization performance evaluation for the TSP. The distance l denotes the smallest distance between two sampling points from two different classes. l is multiplied by four values (0.1, 0.2, 0.3 and 0.4) as the variances of four normal distribution data regions to generate a total of 1940 testing data points in each region.

training points and testing data regions for the evaluation of generalization performance. Around each training point, ten testing data points are randomly generated in a normal distribution. In each region, 1940 testing data points are generated, and this is repeated for ten times. The testing results are listed in Table II. The generalization performance of the MLP-based MPN without weight estimation was 96.85% in average. The average generalization performance for QSNN-based MPN with weight estimation could reach 97.97%, and for the cascade-correlation networks was about 96.25%. In this comparative study, the software packages and the training and testing data as well as the control parameters for good convergence of the cascade-correlation neural network were obtained from the public domain of Carnegie Mellon University (CMU) [9]. In this CMU package, the sigmoidal activation function is used in both the output and hidden units, a pool of eight candidate units, and the maximum learning iteration is limited to 100 for weight-update.

An MPN can be implemented from two different approaches: 1) cost effectiveness (less hardware) or 2) timing (process speed) efficiency. In the cost effective implementation, the DCL engine contains only one MLP (two hidden neurons and one output neuron). The data inputs to the IE first; then, according to the IE outputs C_i, the weights of ith MLP in the DCL engine are selected and applied to an MLP for forward computation. For the efficient timing implementation, all the subnets in the DCL engine are implemented in a parallel structure. The data inputs to all the subnets (MLPs) and the IE, then the output C_i of the IE selects a subnet in the DCL engine for the proper output. In both cases, the computation time ranges from two to four layers of feedforward network processing time.

According to these two different implementations, we can also measure the retrieving time based on multiplication and accumulation steps. This measure leaves out the computation of activation functions, which can be implemented by table look up. The cascade-correlation neural network with 15 hidden nodes needs 152 multiplications and 152 accumulations and the proposed MPN with 22 subnets needs 149 multiplications and 149 accumulations. The performance of the proposed method is slightly better than that of the cascade-correlation neural network based on the sequential approach. If we consider the parallel approach, the retrieving time of the proposed MPN needs only three multiplications and three accumulations, however, the parallel implementation of the cascade-correlation neural network with 15 hidden nodes needs 16 multiplications and 16 accumulations. In other words, the retrieving time can be reduced from 2.0% by sequential or 81.3% by parallel impelementation.

B. Experiments on a Real-World Dataset

The second experiment is performed on a real-world task: to classify the Pima Indians diabetes dataset [22]. The dataset consists of 768 samples taken from patients who may show signs of diabetes. Each sample is described by eight attributes, each attribute has discrete and continuous values. The training set consists of 384 randomly selected samples, and the rest are testing samples. We also use $0.5 * 0.5$ as the squared error tolerance for each training pattern, and perform 30 runs on different randomly sampled training and testing data.

Table III summarizes the learning and testing performance according to the number of epochs, number of presentations, number of subnetworks on the Pima database. We have compared the performance of QSNN-based MPN, MLP-based MPN and a single standtard MLP. The single MLP is composed of an 8–12–1 structure and is trained by backpropagation algorithm. The MLP and QSNN used in MPN are composed of 8–2–1 structure and are also trained by backpropagation algorithm. The MLP based MPN shows that the number of presentation is greatly reduced, and the training and testing performance

TABLE III
PERFORMANCE OF DIFFERENT NEURAL NETWORKS ON THE PIMA DATABASE [22]. NUMBERS IN ( ) INDICATES THE STANDARD DEVIATION

Types of NN	No. of epochs	No. of presentations	No. of subnets subnets	Training accuracy(%)	Testing accuracy (%)
8-12-1 MLP	2835(1118)	1088640(429312)	NA	75.71(4.42)	66.62(4.09)
MPN(MLP)	1136(755)	42639(6604)	8.07(3.46)	78.89(2.83)	73.85(1.95)
MPN(MLP)+WE	541(286)	36292(3957)	5.73(1.34)	78.75(1.37)	74.78(1.89)
MPN(QSNN)	100(119)	19284(1778)	2.33(3.11)	78.33(1.72)	76.36(2.08)
MPN(QSNN)+WE	49(3)	18689(1122)	1.0(0.0)	78.13(2.01)	76.41(1.84)

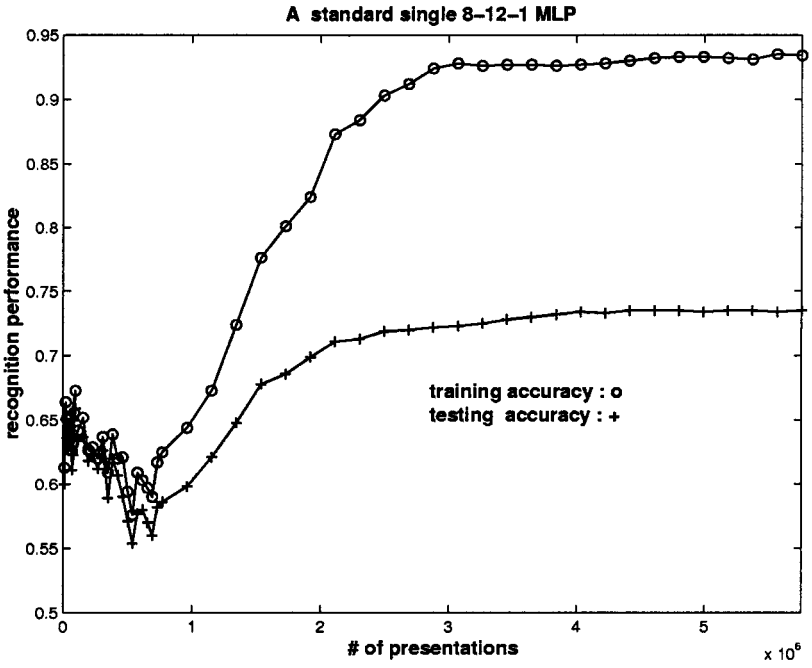

Fig. 12. Recognition performance versus iteration for a standard 8–12–1 single MLP.

are improved 3.2% and 7.2%, respectively. Incorporating with weight estimation, the MPN can further reduce the number of training presentation and subnetworks about 14.9% and 29.0%, respectively. By using QSNN and weight estimation, the MPN requres only one subnet and 49 learning epochs to achieve 78.1% and 76.4% of training and testing performance, respectively. Setiono and Liu [26] achieve a 93.6 (2.77)% and 71.0 (1.74)% training and testing performance. Friedman [12] report their best testing performance to be 76.30 (1.24)% over a various of Bayesian network classifiers. By using several learners with different learning strength, the popular boosting algorithm [11] reports their testing performance around 74.3% $\sim$ 75.6%.

One of the most serious problems in applying flexible machine learning techniques to noisy real-world data is the problem of overfitting. To study the effect of the MPN architecture and the DCL learning scheme on the overfitting, we believe monitoring the training and testing errors can be very helpful. One manifestation of overfitting is when the performance on out-of-sample data, plotted as a function of training time, starts deteriorating after having reached on optimal point. We compare the dynamics of overfitting on the three architectures: a standard single 8–12–1 MLP, an 8–2–1 MLP-based MPN, and an 8–2–1 QSNN-based MPN.

Figs. 12–14 reveal significantly different degrees of overfitting between MPNs and the standard MLP. Whereas the learning of the MPNs is stable and does not overfit much, the standard MLP is somewhat worse. Note that the in-sample (training) error is somewhat lower for the standard MLP than for the other two MPN architectures: the standard MLP is trained to minimize precisely one error function, whereas in the other cases, more error functions are minimized to achieve an averaged performance. Our explanation that MPNs have better antioverfitting capabilities is described as follows. In general, as training proceeds the optimal point, the network tends to shift its resources toward the high noise regions: the more noisy data points, the bigger its error and thus bigger its effect in error backpropagation. We assume that there are some regions in input space that are more noisy than others ("noise heterogeneity"). If every data point is equally presented to the network, the noisy regions tend to attract the resources of the network, mistaking the noise as signal and trying to model it (i.e., overfitting). At the same time, the resources are moved away from the less noisy regions, resulting in underfitting there. Using modular networks and partitioning input space according to the error correlation scheme, usually faring much better than "training until convergence," can alleviate this problem, since independent local learning structure and separated input data

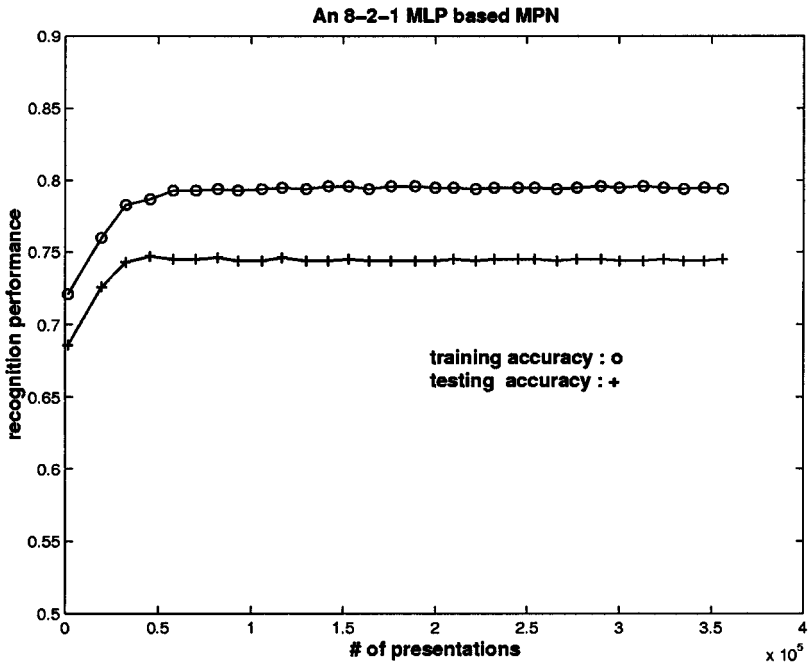

Fig. 13. Recognition performance versus iteration for an 8–12–1 MLP-based MPN.

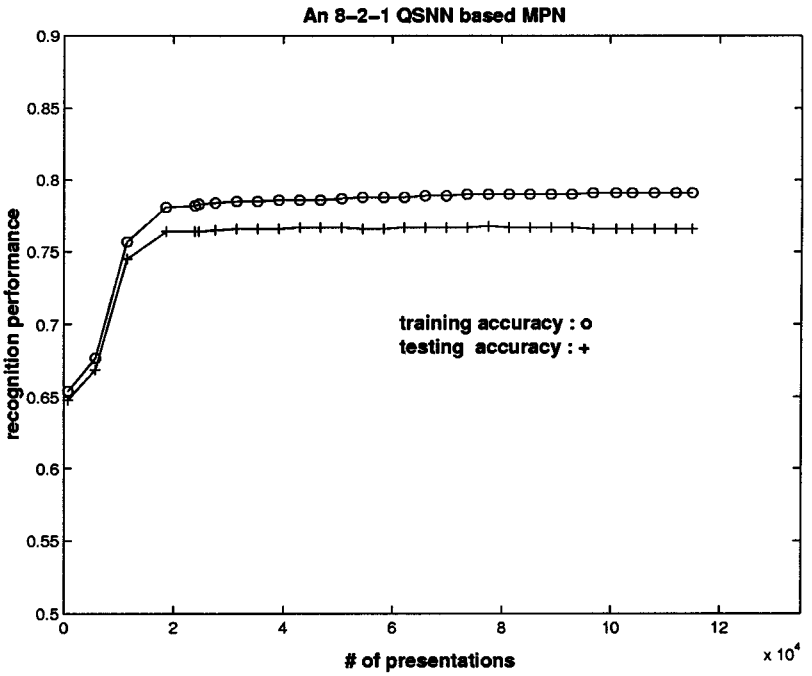

Fig. 14. Recognition performance versus iteration for an 8–12–1 QSNN-based MPN.

regions prevent an MPN modular network from entering into such a global error model.

IV. Concluding Remarks

The modular perceptron network with DCL proposed in this paper is a self-growing modular neural network. Each of the individual subnets in an MPN is a very simple MLP and not powerful enough to learn a given complicated nonlinear problem. By incorporating the DCL scheme with WE, an MPN can generate appropriate number of subnets to quickly and successfully learn some very complicated nonlinear problems. As far as generalization performance is concerned, the MPN can also maintain a fairly good level of correctness. Since we only prescribe general backprop learning schemes and a layered network structure in each subnetwork, various backpropagation type learning algorithms and/or layer structures can be applied to the subnets in the DCL engine for performance enhancement. According to the simulation results obtained from the TSP, we find that the MPN with DCL is very effective in learning complex problems.

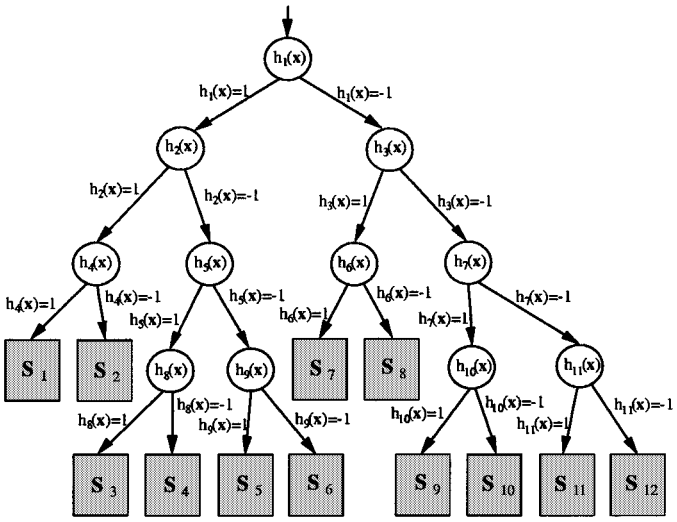

Fig. 15. A decision tree is created to represent the data partition in the DCL engine and the hidden and output nodes in the IE.

It is easy to see that when a new training pattern is added to the original training set, there is only one subnetwork that needs to be retrained. In other words, when a new pattern belongs to one of the partitions of the training set, only the corresponding subnet needs to be retrained. Also, as long as each subnetwork is kept simple and small, the training process can be close to real time. Since all the other partitions, which have been successfully learned, need not be retrained, the MPN is capable of modeling in a dynamically changing environment. Thus, we would like to claim that the MPN is suitable for on-line and incremental learning problems.

APPENDIX
DERIVATION OF WEIGHT VALUES FOR THE INTEGRATION ENGINE

First of all, the process of data region partitioning or the process of subnetwork creation in the DCL engine can be represented as a decision tree structure. As an example, the root node in Fig. 15 represents the first partition process, and the nodes in the following layers represent the subsequent partition processes. In addition, each nonleaf node i in the decision tree corresponds to a hidden neuron i with output $h_i(\boldsymbol{x})$, and each leaf node represents a subset of training data which can be successfully learned by a subnetwork in the DCL engine.

In the following, some properties of the decision tree that relate to the IE are described. There exists a unique forward path from the root node to a leaf node S_i. By concatenating the outputs of the nonleaf (i.e., the hidden) nodes along the path, a unique identification string can be formed, e.g., $I_i(=s_{i1}s_{i2}\ldots s_{id_i})$, $s_{ij} \in \{h_k \bigcup \overline{h}_k | 1 \le i \le N-1\}$, and d_i denotes the path depth from the root to a leaf node S_i. For example as shown in Fig. 15, the leaf node S_5 can be reached along the string $h_1\overline{h}_2\overline{h}_5h_9$, where h_i represents the positive output (1) of the ith hidden neuron and $\overline{h}_i$ indicates the negative output (-1) of the ith hidden neuron. In a decision tree, for any two leaf nodes, there exists one and only one common node h_i with complement output (h_i or $\overline{h}_i$) values in their identification strings. For example, h_9 is the common node, which outputs complementary values h_9 and $\overline{h}_9$ to S_5 and S_6, respectively. In addition, when a decision tree contains N leaf nodes, then there exist $N-1$ internal (hidden) nodes in the tree. As far as data partitioning is concerned, each leaf node corresponds to a data region which has been learned by a subnet (an MLP) in DCL engine. Let the sets H and H^c represents two types of the outputs from the $N-1$ hidden nodes

$$H = \{h_i | 1 \le i \le N-1\}$$

and

$$H^c = \left\{\overline{h}_i | 1 \le i \le N-1\right\}.$$

For an identification string I_i with path depth d_i, if there are m components in H, then there are $d_i - m$ components in H^c. Suppose an input pattern x belongs to the subset S_i; then, the IE will generate a control signal $c_i(x)$ from its ith output neuron S_i. We can formulate this concept as follows:

- **C1:** $c_i(\boldsymbol{x}) = \mathcal{H}_i(u_{i,0} + \sum_{k=1}^{N-1} u_{i,k}h_k(\boldsymbol{x})) = 1$, for $\boldsymbol{x} \in S_i$;
- **C2:** $c_i(\boldsymbol{x}) = \mathcal{H}_i(u_{i,0} + \sum_{k=1}^{N-1} u_{i,k}h_k(\boldsymbol{x})) = 0$, for $\boldsymbol{x} \notin S_i$;

where $\mathcal{H}_i(\,)$ denotes the *Heaviside* activation function of the ith output neuron. In order to simplify the computation of the learning algorithm for the IE, the following value assignment rules for the weights ($u_{i,j}$) are suggested:

- **U1:** $u_{i,j} = v$ if $s_{i,j}$ is in H;
- **U2:** $u_{i,j} = v'$ if $s_{i,j}$ is in H^c;
- **U3:** $u_{i,j} = 0$ (no connection), if h_j is not contained in I_i;

for arbitrary v and v'. Since rules U1, U2, and U3 specify the connection $u_{i,j}$ between hidden neurons and output neurons to be one of the three possible values: $\{v, 0 \text{ and } v'\}$, rules C1 and C2 can be further expressed as follows:

- **C′1:** $\mathcal{H}_i(u_{i,0} + mv - (d_i - m)v') = 1$, for $\boldsymbol{x} \in S_i$;
- **C′2:** $\mathcal{H}_i(u_{i,0} + (m-1)v - v - (d_i - m)v') = 0$, for $\boldsymbol{x} \notin S_i$, and the hidden neuron i in H, or $\mathcal{H}_i(u_{i,0} + mv - (d_i - m - 1)v' + v') = 0$, for $\boldsymbol{x} \notin S_i$, and the hidden neuron i in H^c.

By setting $v > 0$ and $v' < 0$, $(\mathbf{C'1})$ and $(\mathbf{C'2})$ can be rewritten as

$$u_{i,0} + mv - (d_i - m)v' \ge 0 \tag{12}$$

$$u_{i,0} + (m-1)v - (d_i - m)v' - v < 0 \tag{13}$$

$$u_{i,0} + mv + v' - (d_i - m - 1)v' < 0. \tag{14}$$

If we set $v = 1$ and $v' = -1$, the above three relations can be merged as

$$2 - d_i > u_{i,0} \ge -d_i. \tag{15}$$

Therefore, the values of $u_{i,0}$ can be assigned as $-d_i$.

ACKNOWLEDGMENT

The authors would like to thank Prof. S.Y. Kung and Dr. K. I. Diamantaras for their helpful suggestions regarding the OPCA.

REFERENCES

[1] A. Atiya, R. Aiyad, and S. Shaheen, "A practical gated expert network," in *Proc. 1998 IEEE Int. Joint Conf. Neural Network (IJCNN)*, vol. I, AK, May 1998, pp. 419–424.

[2] G. Auda and M. Kamel, "Modular Neural Networks: A survey," *Int. J. Neural Syst.*, vol. 9, no. 2, pp. 129–151, April 1999.

[3] C. C. Chiang and H. C. Fu, "A divide-and-conquer methodology for modular supervised neural network design," in *Int. Conf. Neural Networks*, 1994, pp. 119–124.

[4] ——, "Using multithreshold quadratic sigmoidal neurons to improve classification capability of multilayer perceptrons," *IEEE Trans. Neural Networks*, vol. 5, pp. 516–519, May 1994.

[5] K. I. Diamantaras and S. Y. Kung, "An unsupervised neural model for oriented principal component extraction," in *Proc. Int. Conf. Acoust., Speech, Signal Processing*, Toronto, ON, Canada, 1991, pp. 1049–1052.

[6] H. Drucker, R. Schapire, and P. Simard, "Boosting performance in neural networks," *Int. J. Pattern Recognition Artifical Intell.*, pp. 705–719, 1993.

[7] H. Drucker, R. E. Schapire, and P. Simard, "Improving performance in neural networks using boosting," in *Advances in Neural Information Processing Systems 5*, S. J. Hanson, J. D. Cowan, and C. L. Giles, Eds. San Mateo, CA: Morgan Kaufmann, 1993, pp. 42–49.

[8] H. Drucker and C. Cortes, "Boosting decision trees," in *Advances in Neural Information Processing Systems 8*, D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, Eds. Cambridge, MA: MIT Press, 1996, pp. 479–485.

[9] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1990, pp. 524–532.

[10] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," in *Proc. 2nd European Conf. Comput. Learning Theory*, March 1995, pp. 23–37.

[11] ——, "Experiments with a new boosting algorithm," in *Machine Learning: Proc. 13th Int. Conf.*, 1996, pp. 148–156.

[12] N. Friedman, D. Geiger, and M. Goldsmidt, "Bayesian network classifiers," *Machine Learning*, vol. 29, pp. 131–163, 1997.

[13] S. J. Hanson, "Meiosis networks," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1990, pp. 533–541.

[14] S. Hashem, "Algorithms for optimal linear combinations of neural networks," in *IEEE Int. Conf. Neural Networks(ICNN97)*, vol. 1, Houston, TX, June 9–12, 1997, pp. 242–247.

[15] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1999.

[16] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural Comput.*, vol. 3, pp. 79–87, 1991.

[17] M. I. Jordan and R. A. Jacobs, "Hierarchies mixtures of experts and the EM algorithm," *Neural Comput.*, vol. 6, pp. 181–214, 1994.

[18] N. B. Karayiannis, "Hybrid learning schemes for fast training of feed-forward neural networks," *Math. Comput. Simulation*, vol. 41, pp. 13–28, 1996.

[19] K. K. Lang and M. K. Witbrock, "Learning to tell two spirals apart," in *Proc. 1988 Connectionist Models Summer School*. San Mateo, CA: Morgan Kaufmann, 1988, pp. 52–61.

[20] Y. P. Lee and H. C. Fu, "Weight estimation for the learning of modular perceptron networks," in *Proc. 1999 IEEE Workshop on Neural Networks for Signal Processing IX*, Madison, WI, Aug. 23–25, 1999.

[21] A. J. Maren, C. T. Harston, and R. M. Pap, *Handbook of Neural Computing Applications*. San Diego, CA: Academic, 1990.

[22] P. M. Murphy and D. W. Aha, "UCI repository of machine learning databases,", http://www.ics.uci.edu/~mlearn/MLRepository.html.

[23] U. Naftaly, N. Intrator, and D. Horn, "Optimal ensemble averaging of neural network," *Neural Networks*, vol. 8, pp. 283–296, 1997.

[24] N. J. Nilsson, *Learning Machines: Foundations of Trainable Pattern-Classifying Systems*. New York: McGraw-Hill, 1965.

[25] E. Ronco and P. Gawthrop, "Modular neural networks: A state of the art," Center Syst. Contr, Univ. Glasgow, Glasgow, U.K., Technical report: CSC-95 026, 1995.

[26] R. Setiono and H. Liu, "Neural-network feature selector," *IEEE Trans. Neural Networks*, vol. 8, May 1997.

[27] R. Schapire, "The strength of weak learnability," *Machine Learning*, vol. 5, no. 2, pp. 197–227, 1990.

[28] L. G. Valiant, "A theory of the learnable," *Commun. ACM*, vol. 27, no. 11, pp. 1134–1142, 1984.

[29] M. Wynne-Jones, "Node splitting: A constructive algorithm for feed-forward neural networks," *Neural Computing and Applications*, vol. 1, no. 1, pp. 17–22, 1993.

Hsin-Chia Fu (M'78) received the B.S. degree from National Chiao-Tung University, Taiwan, R.O.C., in electrical and communication engineering in 1972, and the M.S. and Ph.D. degrees from New Mexico State University, Las Cruces, both in electrical and computer engineering in 1975 and 1981, respectively.

From 1981 to 1983, he was a Member of the Technical Staff at Bell Laboratories, Indianapolis, IN. Since 1983, he has been on the faculty of the Department of Computer Science and Information Engineering at National Chiao-Tung University. From 1987 to 1988, he served as the Director of the Department of Information Management, Research Development and Evaluation Commission, Executive Yuan, R.O.C. From 1988 to 1989, he was a Visiting Scholar at Princeton University, Princeton, NJ. From 1989 to 1991, he served as the Chairman of the Department of Computer Science and Information Engineering at National Chiao-Tung University. From September to December of 1994, he was a Visiting Scientist at Fraunhofer-Institut for Production Systems and Design Technology (IPK), Berlin Germany. His research interests include digital signal/image processing, VLSI array processors, and neural networks. He has authored more than 100 technical papers, and two textbooks, *PC/XT BIOS Analysis* (Taipei, Taiwan: Sun-Kung Book Co., 1986), and *Introduction to Neural Networks* (Taipei, Taiwan: Third Wave Publishing Co., 1994).

Dr. Fu was the corecipient of the 1992 and 1993 Long-Term Best Thesis Award with Dr. K.-T. Sun and Dr. C.-C. Chiang, respectively, and the recipient of the 1996 Xerox OA paper Award. He has served as a Founding Member, Program Cochair in 1993 and General Cochair in 1995 of International Symposium on Artificial Neural Networks, and served the Technical Committee on Neural Networks for Signal Processing of the IEEE Signal Processing Society from 1998 to 2000. He is a Member of the IEEE Signal Processing and Computer Societies, Phi Tau Phi, and the Eta Kappa Nu Electrical Engineering Honor Society.

Yen-Po Lee received the B.S. degree in naval architecture engineering and the M.S. degree in mechanical engineering from Chung Cheng Institute of Technology, Taiwan R.O.C, in 1980 and 1984, respectively. He is an Associate Scientist in the Systems Development Center, Chung-Shan Institute of Science and Technology, Taiwan, R.O.C. His research interests include neural networks, data mining, missile guidance design and simulation.

Cheng-Chin Chiang received the B.S. degree in electrical engineering from National Cheng-Kung University, Tainan, Taiwan, R.O.C., in 1986, and the M.S. and Ph.D. degrees in computer science and information engineering from National Chiao-Tung University, Hsinchu, Taiwan, R.O.C., in 1988 and 1993, respectively. From 1993 to 2000, he was a Researcher of Advanced Technology Center of Computer and Communication Research Laboratories in Industrial Technology Research Institute (ITRI), Hsinchu, Taiwan. In August 2000, he joined the faculty of the Department of Computer Science and Information Engineering, National Dong-Hwa University, Hualien, Taiwan. His research interests include neural networks, pattern recognition, multimedia systems, virtual reality and content-based multimedia retrieval. Dr. Chiang received the honor of Long-Term Doctorial Thesis Award from Acer Corporation in 1993, the honor of Outstanding Research Achievements Award from ITRI in 1996 and the honor of Outstanding Young Engineer Award from Chinese Institute of Engineers in 2000.

Hsiao-Tien Pao received the B.S. degree from National Cheng-Kung University, Taiwan, R.O.C., in mathematics in 1976, and the M.S. and Ph.D. degrees from National Chiao-Tung University, Taiwan, R.O.C., both in applied mathematics in 1981 and 1994, respectively.

From 1983 to 1985 she was a Member of the Assistant Technical Staff at Tel-Communication Laboratories, Chung-Li, Taiwan, R.O.C. Since 1985, she has been on the faculty of the Department of Management Science at National Chiao-Tung University, in Taiwan, R.O.C. Her research interests include statistics, and neural networks.