A Constructive Algorithm for Training Cooperative Neural Network Ensembles

Md. Monirul Islam, Xin Yao, *Fellow, IEEE*, and Kazuyuki Murase

***Abstract*—This paper presents a constructive algorithm for training cooperative neural-network ensembles (CNNEs). CNNE combines ensemble architecture design with cooperative training for individual neural networks (NNs) in ensembles. Unlike most previous studies on training ensembles, CNNE puts emphasis on both accuracy and diversity among individual NNs in an ensemble. In order to maintain accuracy among individual NNs, the number of hidden nodes in individual NNs are also determined by a constructive approach. Incremental training based on negative correlation is used in CNNE to train individual NNs for different numbers of training epochs. The use of negative correlation learning and different training epochs for training individual NNs reflect CNNEs emphasis on diversity among individual NNs in an ensemble. CNNE has been tested extensively on a number of benchmark problems in machine learning and neural networks, including Australian credit card assessment, breast cancer, diabetes, glass, heart disease, letter recognition, soybean, and Mackey–Glass time series prediction problems. The experimental results show that CNNE can produce NN ensembles with good generalization ability.**

***Index Terms*—Constructive approach, diversity, generalization, negative correlation learning, neural-network (NN) ensemble design.**

I. Introduction

MANY real-world problems are too large and too complex for a single neural network (NN) to solve alone. An NN ensemble consisting of several individual NNs has been shown to be able to improve NNs generalization performance [48], [49]. The goal of NN ensemble design is to determine ensemble architectures automatically. Architecture selection requires appropriate choice of both the number of individual NNs and their hidden nodes. Both theoretical [17], [26] and empirical studies [18], [41] have demonstrated that the generalization performance of NN ensembles depends greatly on both accuracy and diversity among individual NNs in an ensemble. An automatic approach to ensemble architecture design that considers both accuracy and diversity is clearly needed.

There has been much work in training NN ensembles [46], [48], in mixtures of experts [21], [22], and in various boosting and bagging methods [9]. However, all these methods are used to change weights in an ensemble. The structure of the ensemble, e.g., the number of NNs in the ensemble, and the structure of individual NNs, e.g., the number of hidden nodes, are all designed manually and fixed during the training process. While manual design of NNs and ensembles might be appropriate for problems where rich prior knowledge and an experienced NN expert exist, it often involves a tedious trial-and-error process for many real-world problems because rich prior knowledge and experienced human experts are hard to get in practice.

This paper proposes a new constructive algorithm, called constructive NN ensemble (CNNE), for training cooperative NN ensembles. It is the first algorithm, to our best knowledge, that combines ensemble's architecture design with cooperative training of individual NNs in an ensemble. It determines automatically not only the number of NNs in an ensemble, but also the number of hidden nodes in individual NNs. It uses incremental training based on negative correlation learning [32], [33] in training individual NNs. The main advantage of negative correlation learning is that it encourages different individual NNs to learn different aspects of the training data so that the ensemble can learn the whole training data better. It does not require any manual division of the training data to produce different training sets for different individual NNs in an ensemble.

CNNE differs from previous work in designing and training NN ensembles in a number of ways. First, CNNE emphasizes both accuracy and diversity among individual NNs in an ensemble. In order to produce accurate individual NNs, CNNE automatically determines the number of hidden nodes for individual NNs so that they are sufficiently accurate. Two techniques are used in CNNE for producing and maintaining diversity among individual NNs. They are incremental training based on negative correlation learning [32], [33] and different numbers of training epochs for different NNs. While some previous work acknowledged the importance of maintaining both accuracy and diversity among individual NNs in an ensemble [8], [17], [18], [24], [26], [49], [51], [52], few techniques have actually been developed to achieve them [15], [16]. The most common practice was to select the number of hidden nodes in individual NNs randomly and then trained them independently using the same training set or different training sets. However, random selection of hidden nodes may hurt the accuracy of individual NNs. Use of the same training set for all individual NNs in an ensemble may reduce the diversity among NNs, while different training sets for individual NNs may decrease accuracy of individual NNs. It is very difficult to construct appropriate training sets that maintain a good balance between accuracy and diversity among individual NNs in an ensemble [51].

Second, CNNE incrementally trains each NN in an ensemble for a different number of training epochs, which is determined automatically by its training process. This training approach is

Manuscript received February 21, 2002; revised August 27, 2002.

The authors are with the Department of Human and Artificial Intelligence Systems, Fukui University, Fukui 910-8507, Japan (e-mail: murase@synapse.fuis.fukui-u.ac.jp).

Digital Object Identifier 10.1109/TNN.2003.813832

quite different from existing ones which train all individual NNs for a predefined and fixed number of training epochs (τ). The difficulty of using the existing approach lies in the difficulty of selecting a τ that is suitable for all individual NNs in the ensemble.

Third, the number of individual NNs in an ensemble is often predefined and fixed in previous work. However, the number of NNs in an ensemble has a major impact on the performance of the ensemble. If the number of individual NNs is too small, the ensemble is likely to perform poorly in terms of generalization. If the number of individual NNs is too large, the computational cost of training the ensemble will increase. In some cases, the ensemble performance may even deteriorate in an overly large ensemble. To address this problem, CNNE proposed in this paper determines automatically the number of individual NNs in an ensemble as well as the number of hidden nodes in each NN using a constructive approach.

Fourth, CNNE uses a very simple cost function, i.e., the ensemble error, for determining ensemble architectures. This is quite different from some previous work [26], [27], [41] that divides the cost function into two parts: accuracy and diversity. A major disadvantage of this approach lies in the inherent difficulty in weighing and combining the two factors in one function. This difficulty becomes increasingly prominent as the number of individual NNs in an ensemble increases. Determining ensemble architectures automatically by applying a constructive approach and training individual NNs in an ensemble incrementally by negative correlation learning provide a novel and simple alternative that avoids the problem. The effectiveness of our approach has been demonstrated by the experimental results presented in this paper.

Fifth, CNNE has been tested extensively on a number of benchmark problems, including Australian credit card assessment, breast cancer, diabetes, glass, heart disease, letter recognition, soybean, and Mackey–Glass time series prediction problems. Few NN ensemble algorithms have been tested on a similar range of benchmark problems. In terms of generalization ability, our experimental results have showed clearly that CNNE is better than other ensemble and nonensemble algorithms.

The rest of this paper is organized as follows. Section II discusses various individual NN training algorithms and explains why we adopted the one used in CNNE. Section III describes our CNNE algorithm in details. Section IV presents results of our experimental study. Finally, Section V concludes the paper with a brief summary and a few remarks.

II. Training Methods for Individual NNs in Ensembles

One important issue with any NN ensemble algorithms is to decide how to train individual NNs in an ensemble, since the training strategy influences the diversity and accuracy of individual NNs in the ensemble. There are roughly three major approaches to training individual NNs, i.e., independent training, simultaneous training, and sequential training.

In independent training, each individual NN in the ensemble is trained independently to minimize the error between the target and its output. One major disadvantage of this approach is the lack of interaction among individual NNs during training [32], [33]. Therefore, individual NNs may produce positively correlated outputs that are not useful for improving ensemble's performance.

In sequential training, individual NNs in the ensemble are trained one after another not only to minimize the error between the targets and their outputs, but also to decorrelate their errors from previously trained NNs. In this approach, the connection weights of previously trained NNs are frozen when the current NN is being trained. As a result, training an individual NN in an ensemble sequentially cannot affect the previously trained NNs. The errors of individual NNs are not necessarily negatively correlated [32], [33]. It is known that negative correlation among individual NNs in an ensemble is beneficial for improving ensemble's performance [23], [32], [33].

Recently, Liu and Yao [32], [33] proposed simultaneous training for individual NNs in an ensemble for the same number of training epochs. They introduced an unsupervised penalty term in the error function of each individual NN so that individual NNs in an ensemble can be trained simultaneously and interactively. The advantage of this approach is that it can produce biased individual NNs whose errors tend to be negatively correlated. One disadvantage of this approach is that all individual NNs in an ensemble are concerned with the whole ensemble error. This may cause competitions among individual NNs in the ensemble. In addition, it may be difficult to select an appropriate number of training epochs that is suitable for all individual NNs in the ensemble.

None of the above three training approaches have addressed the issue of ensemble architecture design. The number of individual NNs and/or topologies of individuals NNs are usually designed by hand or through a trial-and-error process.

In this paper, CNNE uses a training algorithm that can be regarded as hybrid sequential and simultaneous training. In our algorithm, individual NNs are trained sequentially. However, the weights in previously trained NNs are not frozen while the current NN is being trained.

III. CNNE

CNNE uses incremental training, in association with negative correlation learning, in determining ensemble architectures. In our incremental training, individual NNs and hidden nodes are added to the ensemble architecture one by one in a constructive fashion during training. In order to facilitate interactions among individual NNs in an ensemble, negative correlation learning is used for training individual NNs.

In comparison with other ensemble training algorithms, the major advantages of CNNE include 1) automatic design of ensemble architectures; 2) maintaining of both diversity and accuracy among individual NNs in an ensemble; and 3) a good generalization ability of constructively learned ensembles. Although CNNE allows only three-layered feedforward NNs with a sigmoid transfer function in its current implementation, this is not an inherent constraint. In fact, any type of NNs (e.g., cascade NNs [10]) can be used for individual NNs in the ensemble.

The major steps of CNNE are summarized in Fig. 1, which are explained further as follows.

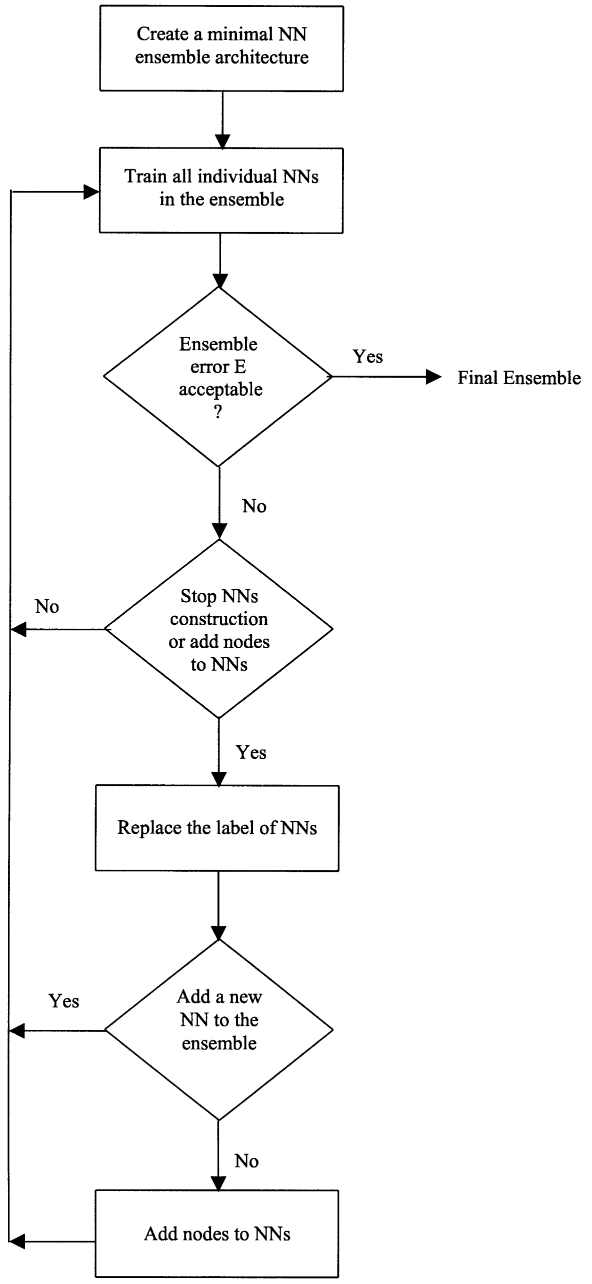

Create a minimal NN ensemble architecture
Train all individual NNs in the ensemble
Ensemble error E acceptable ?
Yes
Final Ensemble
No
Stop NNs construction or add nodes to NNs
No
Yes
Replace the label of NNs
Add a new NN to the ensemble
Yes
No
Add nodes to NNs

Fig. 1. Flowchart of CNNE.

Step 1) Create a minimal ensemble architecture consisting of two individual NNs. Each individual NN has three layers, i.e., an input, an output, and a hidden layer. The number of nodes in the input and output layers is the same as the number of inputs and outputs of the problem. Initially, the hidden layer contains only one node. Randomly initialize all connection weights within a certain small range. Label each individual NN with I.

Step 2) Partially train all individual NNs in the ensemble on the training set for a certain number of training epochs using negative correlation learning. The number of training epochs, τ, is specified by the user. Partial training, which was first used in conjunction with an evolutionary algorithm [53], means that NNs are trained for a fixed number of epochs regardless whether they have converged or not.

Step 3) Compute the ensemble error E on the *validation* set. If E is found unacceptable (i.e., too large), then assume that individual NNs in the ensemble are not trained sufficiently or the ensemble architecture is insufficient, and go to the next step. Otherwise, stop the training process and go to Step 8. In CNNE, the ensemble error E is defined by

$$E = 100\frac{1}{N}\sum_{n=1}^{N}\frac{1}{2}\left(\frac{1}{M}\sum_{i=1}^{M}F_i(n) - d(n)\right)^2 \tag{1}$$

where N is the number of patterns in the validation set, M is the number of individual NNs in the ensemble, $F_i(n)$ is the output of network i on the nth pattern, and $d(n)$ represents the desired output of the nth pattern.

Step 4) Check the criteria for halting network construction and node addition for each NN labeled I. If any NN labeled I does not satisfy the criteria for stopping network construction or node addition, then assume that the I-labeled NNs are not trained sufficiently and go to Step 2 for further training. Otherwise, continue to the next step for the modification of ensemble architecture.

Step 5) Replace the labeling of each I-labeled NN by label F if it satisfies the halting criterion for network construction.

Step 6) If all I-labeled NNs have been replaced by F, add one new NN to the ensemble. Initialize the new NN in the same way as described in Step 1. Label the NN with I and go to Step 2. Otherwise, continue.

Step 7) Add one node to each I-labeled NN that satisfies the criterion for node addition and go to Step 2.

Step 8) Determine the accuracy of the ensemble architecture using the testing set. It should be noted that this is not part of the CNNE training algorithm. It is used for testing the performance of trained ensembles only.

Although the design of NN ensembles could be formulated as a multiobjective optimization problem, CNNE uses a very simple cost function, i.e., the ensemble error. The idea behind CNNE is straightforward. CNNE tries to minimize the ensemble error first by training, then by adding a hidden node to an existing NN, and lastly by adding a new NN. Details about each step of the CNNE algorithm will be given in the following sections.

A. Criteria for Node Addition and Halting Network Construction

CNNE uses simple criteria for deciding when to add hidden nodes in an existing NN and when to halt the construction of individual NNs. The criteria for node addition and halting network construction are based on the same function, i.e., the contribution of an individual NN to the ensemble. The contribution C_i of an individual NN i at any training epoch is

$$C_i = 100\left(\frac{1}{E} - \frac{1}{E^i}\right) \tag{2}$$

where E is the ensemble error *including* individual NN i and E^i is the ensemble error *excluding* individual NN i. The errors are computed according to (1).

According to the node addition criterion, CNNE adds one node to any I-labeled individual NN i when its contribution C_i to the ensemble does not improve by a threshold, ϵ, after a certain number of training epochs. The number of training epochs, τ, is user specified. This criterion is tested for every τ epochs and can be described as

$$C_i(t+\tau) - C_i(t) < \epsilon, \quad t = \tau, 2\tau, 3\tau, \ldots. \tag{3}$$

Another simple criterion is used in CNNE for deciding when to halt the construction of individual NNs. It halts the construction of any I-labeled individual NN i when its contribution C_i to the ensemble, measured after the addition of each hidden node, failed to improve after the addition of a certain number of hidden nodes, indicated by the parameter m_h below. In other words, NN construction stops when the following is true:

$$C_i(m + m_h) \leq C_i(m), \quad m = 1, 2, \ldots \tag{4}$$

where $m_h (m_h > 0)$ is a user specified parameter. If $m_h = 0$, then all individual NNs can consist of one hidden node only. In CNNE, no nodes are added to individual NNs after their construction process have halted.

Although the criteria for node addition and halting network construction in CNNE appear to be similar to those used in some constructive algorithms for monolithic NNs [1], [20], [30], [36], [50], there is a significant difference. Existing constructive algorithms for monolithic NNs use NNs accuracy as their stopping criteria, while CNNE uses NNs *contribution* to an ensemble. There is one major problem of using NNs accuracy as the stopping criterion in the context of ensembles, because more accurate individual NNs do not always lead to an accurate ensemble. Using NNs contribution as a criterion for node addition and halting network construction reflects CNNEs emphasis on *cooperative* training of NNs, because the contribution of any NN in an ensemble could increase only when it would produce accurate but uncorrelated outputs with other NNs in the ensemble.

B. Constructive Approach

CNNE uses a constructive approach to determining both the number of individual NNs and the number of hidden nodes in the NNs. There are a number of reasons why we use a constructive approach in CNNE.

First, it is very difficult to determine the number of individual NNs and each NNs architecture in an ensemble. A constructive algorithm provides an automatic way of designing and training an ensemble without human intervention.

Second, it is relatively easy for an inexperienced user to specify the initial conditions in the constructive approach. For example, the number of individual NNs in an ensemble can simply be set to two and the number of hidden nodes in individual NNs can be just one.

Third, the strong convergence of a constructive algorithm follows directly from its universal approximation ability [28]. Such a result lays a solid theoretical foundation of our algorithm so that we know the ensemble is approximating the right function we are after.

Fourth, a constructive algorithm is computationally more efficient because it always searches small solutions first [29], [30]. Because of smaller solutions, the ensemble is less likely to overfit the training data and, thus, more likely to generalize better.

Fifth, a constructive approach usually requires a relatively small number of user specified parameters. It is known that an important property required for efficient design is an autonomous function independent from, for example, user specified parameters [4]. The use of many user specified parameters requires a user to know rich prior knowledge, which often does not exist for complex real-world problems. The number of user specified parameters used in CNNE is relatively small, i.e., only three (ϵ, τ and m_h), in comparison with other ensemble learning algorithms, e.g., [31].

The parameter τ in CNNE determines when to add a new hidden node in an NN. This parameter is not very critical in CNNE. If τ is small, individual NNs may be under-trained, but CNNE does not freeze the training of individual NNs until the required ensemble error has been reached. If τ is large, individual NNs in an ensemble would be over-trained. However, over-training is not a major problem in ensembles (at least for noiseless training examples). In any case, CNNE deals with this problem by using a validation set to measure the ensemble error.

The other parameter m_h determines when to halt the construction of individual NNs. This parameter is not very critical in CNNE either. For example, if the value of m_h is too large, the number of hidden nodes in individual NNs would be large. In such a case, the number of individual NNs in the ensemble would be small. On the other hand, if the value of m_h is too small, then CNNE may halt the construction of individual NNs very quickly. In such a case, the number of hidden nodes in individual NNs would be small. To offset the effect of early halting, CNNE would add more individual NNs to the ensemble.

C. Architecture Modifications

In CNNE, only when the criteria for node or NN addition are met, will architecture modifications will take place. For architecture modifications, node additions are always attempted before NN additions. CNNE maintains diversity among individual NNs primarily by using negative correlation training. Hidden nodes are added to individual NNs in a constructive fashion to improve the ensemble accuracy. Network additions to an ensemble will be attempted only after adding a certain number of hidden nodes to individual NNs has failed to reduce the ensemble error significantly. Such an ordering of preferring training to adding hidden nodes in existing NNs and preferring growing the existing NNs to adding new NNs bears certain similarity to the previous work on the order of applying architectural mutation in evolving NNs [53].

Node Addition: In CNNE, a hidden node is added by splitting an existing node of the NN. The process of a node splitting is called "cell division" by Odri *et al.* [39]. Two nodes created by splitting an existing node have the same number of connections as the existing node. The weights of the new nodes are calculated according to Odri *et al.* [39]

$$w_{ij}{}^1 = (1+\beta)w_{ij} \tag{5}$$
$$w_{ij}{}^2 = -\beta w_{ij} \tag{6}$$

where w represents the weight vector of the existing node, and w^1 and w^2 are the weight vectors of the new nodes. β is a mutation parameter whose value may be either a fixed or random value. The advantage of this addition process is that it does not require random initialization of the weight vector of the newly added node. As a result, the new NN can maintain the behavioral link with its predecessor better [53].

Network Addition: When the existing ensemble architecture is not capable of reducing the ensemble error below a predefined level and the construction of all individual NNs in the ensemble has been halted, CNNE adds one new NN to the ensemble. The new NN is initialized with one hidden node and small random connection weights. The new NN will be labeled I and trained partially for a fixed number of training epochs using negative correlation learning.

D. Negative Correlation Learning

This section introduces the basic ideas of negative correlation learning. More details can be found in previous publications [32], [33].

Given a training set T of size N

$$T = \{(\mathbf{x}(1), d(1)),\ (\mathbf{x}(2), d(2)),\ \ldots,(\mathbf{x}(N), d(N))\}$$

where $\mathbf{x}$ ($\mathbf{x} \in R^p$) is the input to an NN, d is the desired output and is a scalar. The assumption that d is a scalar is made merely to simplify exposition of the ideas without loss of generality. Consider estimating d by forming an ensemble whose output is a simple average of a set of M NN outputs

$$F(n) = \frac{1}{M}\sum_{i=1}^{M} F_i(n) \tag{7}$$

where $F_i(n)$ and $F(n)$ are the output of the ith individual NN and the ensemble, respectively, on the nth training pattern.

The aim of negative correlation learning is to produce diverse individual NNs in an ensemble. To promote diversity, negative correlation learning introduces a correlation penalty term into the error function of each individual NN in the ensemble. All individual NNs in the ensemble are trained simultaneously and interactively on the same training dataset T. The error function E_i for the ith NN in negative correlation learning is defined by the following equation:

$$\begin{aligned} E_i &= \frac{1}{N}\sum_{n=1}^{N} E_i(n) \\ &= \frac{1}{N}\sum_{n=1}^{N}\left(\frac{1}{2}(F_i(n)-d(n))^2 + \lambda p_i(n)\right) \end{aligned} \tag{8}$$

where $E_i(n)$ is the error of the ith NN after the presentation of the nth training pattern. The first term in (8) is the empirical risk function of the ith NN. The second term p_i is the correlation penalty function. The purpose of minimizing p_i is to penalize positive correlation of errors from different NNs, i.e., to encourage negative correlation of an NNs error with the error of the rest of the ensemble. The parameter $0 \le \lambda \le 1$ is used to adjust the strength of the penalty. The penalty function p_i may use the following equation:

$$p_i(n) = (F_i(n) - F(n))\sum_{j\neq i}(F_j(n) - F(n)). \tag{9}$$

The partial derivative of $E_i(n)$ with respect to the output of network i on the nth training pattern is

$$\begin{aligned} \frac{\partial E_i(n)}{\partial F_i(n)} &= F_i(n) - d(n) + \lambda\frac{\partial p_i(n)}{\partial F_i(n)} \\ &= F_i(n) - d(n) - 2\lambda(F_i(n) - F(n))\left(1 - \frac{1}{M}\right) \\ &\approx F_i(n) - d(n) - 2\lambda(F_i(n) - F(n)) \text{ when } M \text{ is large} \\ &= (1-2\lambda)(F_i(n) - d(n)) + 2\lambda(F(n) - d(n)). \end{aligned} \tag{10}$$

The classical backpropagation (BP) algorithm [47] can be used for weight adjustment in the mode of pattern-by-pattern updating. It can be seen from (10) that negative correlation learning is a simple extension to the classical BP. In fact, the only modification is the calculation of an extra term of the form $2\lambda(F_i(n) - F(n))$ for the ith NN.

The following observations can be made from (8), (9), and (10).

1) During the training process, all individual NNs interact with each other through their penalty terms in the error functions. Each NN minimizes not only the difference between $F_i(n)$ and $d(n)$, but also the difference between $F(n)$ and $d(n)$. In other words, negative correlation learning considers the error that all other NNs have while training an NN.
2) For $\lambda = 0$, there are no correlation penalty terms in the error functions of individual NNs. Individual NNs are just trained independently. That is, independent training is a special case of negative correlation learning.
3) For $\lambda = 1$, we get from (8)

$$\frac{\partial E_i(n)}{\partial F_i(n)} = 2(F(n) - d(n)). \tag{11}$$

Note that the empirical risk function of the ensemble for the nth training pattern is

$$E_{\text{ens}}(n) = \frac{1}{2}\left(\frac{1}{M}\sum_{i=1}^{M} F_i(n) - d(n)\right)^2 \tag{12}$$

where M indicates the number of NNs in the ensemble.

The partial derivative of $E_{\text{ens}}(n)$ with respect to $F_i(n)$ on the nth training pattern is

$$\begin{aligned} \frac{\partial E_{\text{ens}}(n)}{\partial F_i(n)} &= \frac{1}{M}\left(\frac{1}{M}\sum_{j=1}^{M} F_j(n) - d(n)\right) \\ &= \frac{1}{M}(F(n) - d(n)). \end{aligned} \tag{13}$$

In this case, we get

$$\frac{\partial E_i(n)}{\partial F_i(n)} = 2M\frac{\partial E_{\text{ens}}(n)}{\partial F_i(n)}. \tag{14}$$

In other words, the minimization of empirical risk function of the ensemble can be achieved by minimizing the error functions of individual NNs. In effect, a large and more complex task of training the ensemble is automatically decomposed into a number of simpler tasks of training individual NNs.

Because individual NNs in CNNE are trained by negative correlation, different NNs will have different importance for different inputs although (1) and (7) seem to indicate that all individual NNs in the ensemble have the same importance. In any case, it is possible to extend the current work to include other combination strategies that consider the importance of individual NNs explicitly in the ensemble, similar to what we did previously in evolving NN ensembles [54]. Other methods, such as the BKS method [25], genetic-algorithm-based selective ensemble [55] and Bayesian method [3], could also be used.

IV. Experimental Studies

This section evaluates CNNEs performance on several well-known benchmark problems, including classification and time series prediction problems. These problems have been the subject of many studies in NNs and machine learning. Experimental details, results, and comparisons with other work are described.

A. Classification Problems

Seven classification problems are used to evaluate the performance of CNNE. They are the Australian credit card assessment problem, the breast cancer problem, the diabetes problem, the glass problem, the heart disease problem, the letter recognition problem, and the soybean problem. The datasets representing these problems were obtained from the UCI machine learning benchmark repository and were real-world data. The characteristics of the datasets are summarized in Table I. The detailed descriptions of datasets are available at ics.uci.edu (128.195.11) in directory /pub/machine-learning-databases.

1) Experimental Setup: In the past, there was some criticism toward the neural networks benchmarking methodology [42]–[44]. Suggestions for improvement have been put forward [42]–[44]. We follow these suggestions in this paper. All datasets are partitioned into three sets: a training set, a validation set, and a testing set. The testing set is used to evaluate the generalization performance of the trained ensemble and is not seen by any individual NNs during the whole training process.

It is known that the experimental results may vary significantly for different partitions of the same data collection, even when the number of examples in each set is the same [42]. It is necessary to know precise specification of the partition in order to replicate an experiment or conduct fair comparisons. In the following experiments, we partitioned each dataset as follows:

- For the Australian credit card dataset, the first 345 examples were used for the training set, the following 173 examples for the validation set, and the final 172 examples for the testing set.

TABLE I
Characteristics of Classification Datasets

Data set	Number of		
	examples	input attributes	output classes
Australian credit card	690	14	2
Breast cancer	699	9	2
Diabetes	768	8	2
Glass	214	9	6
Heart disease	303	13	2
Letter recognition	20,000	16	26
Soybean	683	35	19

- For the breast cancer dataset, the first 349 examples were used for the training set, the following 175 examples for the validation set, and the final 175 examples for the testing set.
- For the diabetes dataset, the first 384 examples were used for the training set, the following 192 examples for the validation set, and the final 192 examples for the testing set.
- For the glass dataset, the first 107 examples were used for the training set, the following 54 examples for the validation set, and the final 53 examples for the testing set.
- For the heart disease dataset, the first 134 examples were used for the training set, the following 68 examples for the validation set, and the final 68 examples for the testing set. There were 303 examples in the original dataset (Table I). To facilitate comparison with previous work, 33 of them were not used in our experiments due to missing values, etc. So the same dataset was used as in other work [53].
- For the letter recognition dataset, 16 000 and 2000 examples were randomly selected from 20 000 examples for the training and validation sets, and the remaining 2000 examples were used for the testing set.
- For the soybean dataset, the first 342 examples were used for the training set, the following 171 examples for the validation set, and the final 170 examples for the testing set.

It should be kept in mind that such partitions do not represent the optimal ones in practice.

The input attributes of the Australian credit card, diabetes, heart disease, letter recognition and soybean datasets were rescaled to between 0.0 and 1.0 by a linear function. The outputs were encoded by the 1-of-c representation for c classes. The most commonly used winner-takes-all method was used for selecting the NN output. In this method, the output node with the highest activation was designated as the NN output. For each individual NN in an ensemble, one bias node with a fixed input of 1 was used for the hidden and output layers. The

TABLE II
ARCHITECTURES AND ACCURACIES OF TRAINED NN ENSEMBLES FOR THE AUSTRALIAN CREDIT CARD ASSESSMENT PROBLEM FOR DIFFERENT VALUES OF τ AND m_h. THE RESULTS WERE AVERAGED OVER 30 INDEPENDENT RUNS

		Number of networks in the ensemble	Number of hidden nodes in a network	Testing error rate
	Mean	6.5	5.3	0.090
$\tau = 10$	SD	1.5	1.2	0.011
$m_h = 4$	Min	4.0	5.0	0.069
	Max	9.0	8.0	0.116
	Mean	7.8	4.7	0.092
$\tau = 10$	SD	1.7	2.1	0.014
$m_h = 2$	Min	6.0	3.0	0.075
	Max	11.0	7.0	0.133
	Mean	7.4	4.3	0.091
$\tau = 15$	SD	2.0	1.3	0.020
$m_h = 2$	Min	5.0	3.0	0.075
	Max	10.0	7.0	0.127

hidden and output node functions were defined by the logistic sigmoid function $\phi(y) = 1/(1 + \exp(-y))$.

Initial connection weights for individual NNs in an ensemble were randomly chosen in the range between -0.5 and 0.5. The learning rate and momentum for training individual NNs were chosen in the range of 0.10–0.50 and 0.5–0.9, respectively. The number of training epochs τ for partial training of individual NNs was chosen between 5 and 25. The number of hidden nodes m_h used for halting the construction of individual NNs was chosen between one and five. The threshold value ϵ was chosen between 0.10 and 0.20. These parameters were chosen after some preliminary experiments. They were not meant to be optimal. The parameter λ used to adjust the strength of the penalty term was set to 1.0.

2) Experimental Results: Tables II–VIII show the results of CNNE over 30 independent runs on seven different problems. The error rate in the tables refers to the percentage of wrong classification produced by the trained ensemble on the testing set.

It can be observed from Tables II–VIII that the ensemble architectures learned by CNNE were influenced by the values of user specified parameters τ and m_h. For example, for the Australian credit card assessment problem (Table II), when $\tau = 10$ and $m_h = 4$ the average number of individual NNs and hidden nodes were 6.5 and 5.3, respectively, while the average number of individual NNs and hidden nodes were 7.8 and 4.7, respectively, when $\tau = 10$ and $m_h = 2$. This indicates that, for the same value of τ, the number of individual NNs in an ensemble increases when the number of hidden nodes in NNs decreases. This is reasonable because a small NN has only a limited processing power. CNNE added more NNs to the ensemble when the size of individual NNs was small. However, it is worth noting that the testing error rate remained roughly the same for dif-

TABLE III
ARCHITECTURES AND ACCURACIES OF TRAINED NN ENSEMBLES FOR THE CANCER PROBLEM FOR DIFFERENT VALUES OF τ AND m_h. THE RESULTS WERE AVERAGED OVER 30 INDEPENDENT RUNS

		Number of networks in the ensemble	Number of hidden nodes in a network	Testing error rate
	Mean	3.9	3.6	0.015
$\tau = 10$	SD	1.3	1.1	0.007
$m_h = 4$	Min	3.0	1.0	0.000
	Max	5.0	5.0	0.017
	Mean	4.8	2.9	0.013
$\tau = 10$	SD	1.9	1.0	0.005
$m_h = 2$	Min	4.0	2.0	0.000
	Max	7.0	4.0	0.017
	Mean	4.5	2.5	0.012
$\tau = 15$	SD	1.5	0.9	0.006
$m_h = 2$	Min	3.0	1.0	0.000
	Max	7.0	4.0	0.022

TABLE IV
ARCHITECTURES AND ACCURACIES OF TRAINED NN ENSEMBLES FOR THE DIABETES PROBLEM FOR DIFFERENT VALUES OF τ AND m_h. THE RESULTS WERE AVERAGED OVER 30 INDEPENDENT RUNS

		Number of networks in the ensemble	Number of hidden nodes in a network	Testing error rate
	Mean	4.7	4.5	0.2010
$\tau = 10$	SD	1.7	1.5	0.0014
$m_h = 4$	Min	3.0	3.0	0.1822
	Max	7.0	6.0	0.2291
	Mean	6.5	3.4	0.1980
$\tau = 10$	SD	2.0	1.2	0.0013
$m_h = 2$	Min	5.0	2.0	0.1718
	Max	9.0	5.0	0.2187
	Mean	6.2	3.2	0.1960
$\tau = 15$	SD	1.3	1.4	0.0015
$m_h = 2$	Min	4.0	2.0	0.1770
	Max	8.0	5.0	0.2135

TABLE V
ARCHITECTURES AND ACCURACIES OF TRAINED NN ENSEMBLES FOR THE GLASS PROBLEM FOR DIFFERENT VALUES OF τ AND m_h. THE RESULTS WERE AVERAGED OVER 30 INDEPENDENT RUNS

		Number of networks in the ensemble	Number of hidden nodes in a network	Testing error rate
	Mean	4.9	4.6	0.2614
$\tau = 10$	SD	1.6	1.3	0.2014
$m_h = 4$	Min	3.0	3.0	0.2264
	Max	7.0	7.0	0.3018
	Mean	6.2	3.8	0.2680
$\tau = 10$	SD	1.9	1.1	0.2318
$m_h = 2$	Min	5.0	2.0	0.2452
	Max	9.0	6.0	0.3018
	Mean	6.0	3.5	0.2518
$\tau = 15$	SD	1.4	1.5	0.2412
$m_h = 2$	Min	5.0	2.0	0.2075
	Max	9.0	5.0	0.2830

TABLE VI
ARCHITECTURES AND ACCURACIES OF TRAINED NN ENSEMBLES FOR THE HEART DISEASE PROBLEM FOR DIFFERENT VALUES OF τ AND m_h. THE RESULTS WERE AVERAGED OVER 30 INDEPENDENT RUNS

		Number of networks in the ensemble	Number of hidden nodes in a network	Testing Error rate
	Mean	4.6	6.5	0.1401
$\tau = 10$	SD	1.7	1.4	0.0150
$m_h = 4$	Min	2.0	4.0	0.1176
	Max	6.0	8.0	0.1911
	Mean	5.5	4.9	0.1340
$\tau = 10$	SD	1.4	1.3	0.0112
$m_h = 2$	Min	3.0	3.0	0.1176
	Max	7.0	7.0	0.1764
	Mean	5.8	4.2	0.1384
$\tau = 15$	SD	1.9	2.0	0.0121
$m_h = 2$	Min	3.0	3.0	0.1029
	Max	8.0	6.0	0.1911

TABLE VII
ARCHITECTURES AND ACCURACIES OF TRAINED NN ENSEMBLES FOR THE LETTER RECOGNITION PROBLEM FOR DIFFERENT VALUES OF τ AND m_h. THE RESULTS WERE AVERAGED OVER 30 INDEPENDENT RUNS

		Number of networks in the ensemble	Number of hidden nodes in a network	Testing error rate
	Mean	11.6	10.6	0.067
$\tau = 10$	SD	1.9	2.3	1.5
$m_h = 4$	Min	7.0	7.0	0.052
	Max	19.0	20.0	0.117
	Mean	15.3	8.5	0.062
$\tau = 10$	SD	2.4	1.8	1.3
$m_h = 2$	Min	9.0	6.0	0.050
	Max	24.0	15.0	0.090
	Mean	13.9	8.1	0.060
$\tau = 15$	SD	2.3	2.1	1.2
$m_h = 2$	Min	7.0	5.0	0.049
	Max	21.0	13.0	0.088

TABLE VIII
ARCHITECTURES AND ACCURACIES OF TRAINED NN ENSEMBLES FOR THE SOYBEAN PROBLEM FOR DIFFERENT VALUES OF τ AND m_h. THE RESULTS WERE AVERAGED OVER 30 INDEPENDENT RUNS

		Number of networks in the ensemble	Number of hidden nodes in a network	Testing error rate
	Mean	5.3	5.5	0.0812
$\tau = 10$	SD	1.3	1.1	1.0134
$m_h = 4$	Min	4.0	4.0	0.0705
	Max	9.0	7.0	0.1058
	Mean	7.1	4.2	0.0760
$\tau = 10$	SD	1.7	1.9	1.0318
$m_h = 2$	Min	5.0	3.0	0.0647
	Max	10.0	7.0	0.0941
	Mean	6.8	3.8	0.0781
$\tau = 15$	SD	2.0	1.3	1.0412
$m_h = 2$	Min	4.0	3.0	0.0588
	Max	11.0	6.0	0.1000

ferent parameter settings and different ensemble architectures. The choice of different parameters did not affect the performance of the learned ensembles much, which is a highly desirable feature for any NN training algorithm.

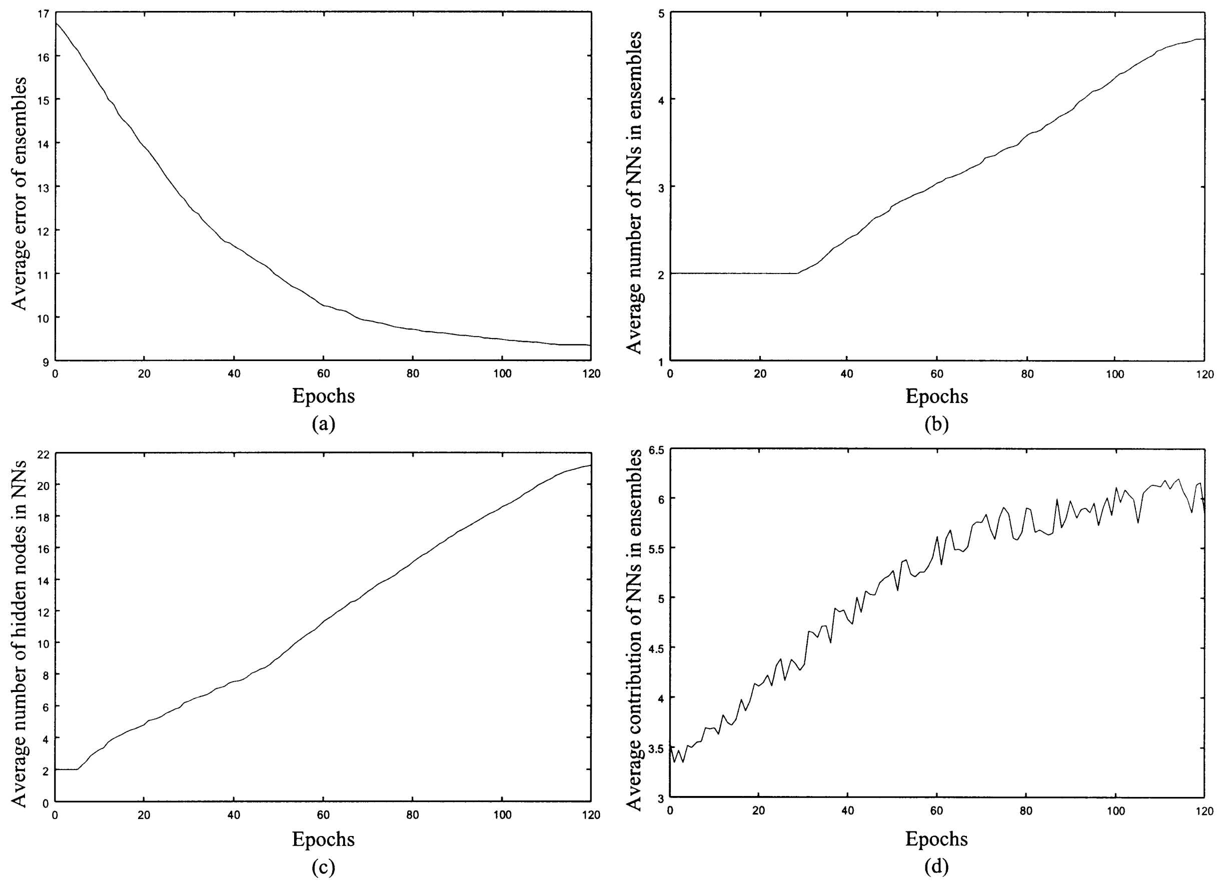


Fig. 2. Training process of CNNE for the diabetes problem.

In some cases, CNNE tended to produce slightly more compact architectures when τ was large (given the same m_h) because a large τ allowed individual NNs in an ensemble to have more training after adding hidden nodes to NNs. For the cancer problem (Table III), for example, when $\tau = 10$ and $m_h = 2$ the average number of individual NNs and hidden nodes were 4.8 and 2.9, respectively, while the average number of individual NNs and hidden nodes were 4.5 and 2.5, respectively, when $\tau = 15$ and $m_h = 2$. Again, the generalization performance of CNNE was not affected much by such variations of parameters and architectures.

CNNEs ability of constructing different ensembles for different problems automatically can be seen clearly from the tables. CNNE produced large ensembles for the letter recognition problem (Table VII), which is large in comparison with other problems we have, and smaller ensembles for other problems. In terms of average results (Table VII), for $\tau = 10$ and $m_h = 2$, CNNE produced ensembles that had 15.3 individual NNs with 8.5 hidden nodes for the letter recognition problem, while it produced ensembles that had only 5.5 individual NNs with 4.9 hidden nodes for the heart disease problem (Table VI). The number of training examples for the letter recognition problem was 16 000, while it was 134 for the heart disease problem. It is reasonable to have a larger ensemble process a large number of training examples. However, there are other factors in addition to the size of training sets, e.g., complexity of the given problem and noise in the training set, that influence the ensemble architecture. For example, the number of training examples for the diabetes problem (Table IV) was 384, while it was 342 for the soybean problem (Table VIII). In terms of average results for $\tau = 10$ and $m_h = 4$, CNNE produced ensembles that had 4.7 individual NNs with 4.5 hidden nodes for the diabetes problem (Table IV), while it produced ensembles that had 5.3 individual NNs with 5.5 hidden nodes for the soybean problem (Table VIII). In general, all the above examples illustrated the same point, i.e., CNNEs ability in determining the ensemble automatically for different problems without human intervention.

To observe the training process in CNNE, Fig. 2 shows, as the number of training epochs increases, the average error of the ensemble, the average number of NNs in the ensemble, the average number of hidden nodes in an NN, and the average contribution of NNs for the diabetes problem. Several observations can be made from the experimental results summarized in the figure and previous given tables.

First, CNNE was capable of finding a satisfactory ensemble architecture through the constructive process. It could dynamically add NNs to an ensemble and hidden nodes to NNs during training, depending on whether the ensemble had insufficient number of NNs or whether the NNs had insufficient number of hidden nodes. For example, the initial number of NNs and their hidden nodes in an ensemble were two and one, respectively, which were small. CNNE first added a few nodes to existing NNs in an ensemble. When adding hidden nodes had not achieved the minimum ensemble error, CNNE started adding new NNs to the ensemble.

Second, CNNE added different NNs to an ensemble at different training epochs during training [Fig. 2(b)]. The training of different NNs started at different times, similar to the case of sequential training.

Third, although adding hidden nodes and further training always improved ensemble's accuracy [Fig. 2(a)], they did not always improve the contribution of NNs. The reason is that, as mentioned previously, the contribution of NNs in an ensemble would increase only when it would produce accurate as well as uncorrelated or negative correlated output with other NNs in the ensemble.

Fourth, the fluctuation of NNs contribution [Fig. 2(d)] indicated the necessity for continuous training of NNs in an ensemble. For example, if one intends to freeze the weights of an NN based on some criteria, such as those used in the sequential training method, the contribution of the frozen NN to the ensemble may decrease as the training process of the ensemble progresses.

Fifth, although the initial ensemble architecture had the same number (i.e., two) of NNs and hidden nodes and CNNE added one NN or one hidden node in each time step, the final ensemble architecture for the diabetes problem had on average about 4.5 NNs and 21.0 hidden nodes. This indicated that hidden nodes were added more frequently than NNs. While the diversity among NNs in an ensemble was maintained by negative correlation learning, adding hidden nodes would improve ensemble's accuracy. Both accuracy and diversity were considered automatically in CNNE.

3) Accuracy and Diversity: This subsection analyzes ensemble's accuracy and diversity in order to gain a better understanding of the CNNE algorithm. The correct response sets of individual NNs are used to determine the accuracy of individual NNs in an ensemble. The diversity ψ determines how many different examples are correctly classified by individual NNs in an ensemble. Let S_i be the correct response set of individual NN i in the testing set, which consists of all the patterns which are correctly classified by NN i, ϕ_i be the size of S_i, and $\phi_{i_1 i_2 \ldots i_k}$ be the size of set $S_{i_1} \cap S_{i_2} \cdots \cap S_{i_k}$. Then ψ among individual NNs in an ensemble is defined as $\phi_{\min} - \phi_{i_1 i_2 \ldots i_k}$, where $\phi_{\min}$ is the size of $S_{\min}$ for the least accurate NN among all NNs in the ensemble.

For the Australian credit card assessment problem, a typical ensemble produced by CNNE among the 30 runs had four networks N_1, N_2, N_3, and N_4. The sizes of correct response sets S_1, S_2, S_3, and S_4 were 159, 150, 125, and 105, respectively. The large variation among networks' accuracies were due to the incremental approach adopted in CNNE for training the ensemble. For example, the initial ensemble consisted of networks N_1 and N_2 only. CNNE first trained N_1 and N_2. When these two NNs had not achieved the required ensemble error, CNNE added N_3 to the ensemble and trained N_3 incrementally. Finally, CNNE added N_4 and trained it incrementally. The size of $S_1 \cap S_2 \cap S_3 \cap S_4$ was only 57. So ψ among networks N_1, N_2, N_3, and N_4 was 48 $(= 105 - 57)$.

In contrast, although the four individual NNs created by independent and simultaneous training reported in [32] had very similar accuracies, the diversity among them was much poorer. The sizes of correct response sets S_1, S_2, S_3, and S_4 for independent and simultaneous training were in the range of 146–149 and 138–147, respectively. The sizes of their intersection sets $S_1 \cap S_2 \cap S_3 \cap S_4$ were 143 and 113, respectively, for independent and simultaneous training. Hence ψs of independent and simultaneous training were 4 $(= 147 - 143)$ and 25 $(= 138 - 113)$, respectively. It is clear that CNNE can produce much diverse NNs in an ensemble in comparison with independent and simultaneous training. Table IX summarizes the above results.

TABLE IX
ACCURACIES AND DIVERSITY OF FOUR INDIVIDUAL NETWORKS TRAINED BY CNNE, INDEPENDENT TRAINING METHOD, AND SIMULTANEOUS TRAINING METHOD FOR THE AUSTRALIAN CREDIT CARD ASSESSMENT PROBLEM. ϕ_i REPRESENTS THE ACCURACY OF THE INDIVIDUAL NETWORK i ON THE TESTING SET. ψ REPRESENTS THE DIVERSITY AMONG FOUR INDIVIDUAL NETWORKS ON THE TESTING SET. ^{b} REPORTED IN [32]

Method	Accuracy				Diversity (ψ)
	ϕ_1	ϕ_2	ϕ_3	ϕ_4	
Incremental	159	150	125	105	48
Independent^b	149	147	148	148	4
Simultaneous^b	147	143	138	143	25

4) Comparison With Other Work: This section compares experimental results of CNNE (for $\tau = 10$ and $m_h = 2$) with those of acasper [38], EPNet [53], CELS [33], EENCL [31], and three other ensemble training algorithms, i.e., bagging [6], arc-boosting [7], and ada-boosting [14], tested by Opitz and Maclin [40]. acasper and EPNet are single NN design algorithms, while CELS, EENCL, bagging, arc-boosting, and ada-boosting are NN ensemble training algorithms. Although there are other ensemble algorithms, such as ensembles of decision trees and others [2], [8], [45], CNNE is only compared with other NN ensemble algorithms here. The primary aim of this paper is not to exhaustively compare CNNE with all other ensemble algorithms, but to evaluate CNNE in order to gain a deeper understanding of automatic ensemble learning. The reason for choosing two nonensemble algorithms is to gain some insights into when and why NN ensembles may be better than single NNs.

The average results of CNNE and EPNet were obtained by averaging over 30 independent runs. The results for acasper and CELS were obtained by averaging over 50 and 25 runs, respectively. The average results of EENCL were obtained by tenfold cross-validation for the Australian credit card assessment problem, and by twelve-fold cross-validation for the diabetes problem. Opitz and Maclin [40] also used five standard tenfold cross-validation for bagging, arc-boosting, and ada-boosting. For each tenfold cross-validation, they first partitioned the original dataset into ten equal-sized sets, then each set was in turn used as the test set while the remaining nine sets were used for training individual NNs in an ensemble [40].

Australian Credit Card Assessment Problem: Table X compares CNNEs results with those produced by EPNet [53], CELS [33], and three other ensemble algorithms tested by Opitz and Maclin [40]. CNNE was able to achieve the smallest average testing error rate among all algorithms compared. For example, the average testing error rate achieved by CELS was 0.120, while it was only 0.092 for CNNE.

TABLE X
COMPARISON AMONG CNNE, EPNET [53], EENCL [31], CELS [33], AND THREE OTHER ENSEMBLE ALGORITHMS [40] IN TERMS OF THE AVERAGE TESTING ERROR RATE (TER) ON THE AUSTRALIAN CREDIT CARD ASSESSMENT PROBLEM. THE RESULTS OF CNNE AND CELS WERE AVERAGED OVER 30 AND 25 RUNS, RESPECTIVELY. THE RESULTS OF EENCL AND OTHERS WERE OBTAINED BY TWELVEFOLD AND FIVE STANDARD TENFOLD CROSS-VALIDATION

	CNNE	EPNet	Bagging	Arc-boosting	Ada-boosting	CELS	EENCL
TER	0.092	0.115	0.138	0.158	0.157	0.120	0.132

TABLE XI
COMPARISON AMONG CNNE, EPNET [53], AND THREE OTHER ENSEMBLE ALGORITHMS [40] IN TERMS OF THE AVERAGE TER FOR THE CANCER PROBLEM. THE RESULTS OF CNNE AND EPNET WERE AVERAGED OVER 30 RUNS, WHILE THEY WERE AVERAGED OVER FIVE STANDARD TENFOLD CROSS VALIDATIONS BY OTHER ALGORITHMS

	CNNE	EPNet	Bagging	Arc-boosting	Ada-boosting
TER	0.013	0.014	0.034	0.038	0.040

TABLE XII
COMPARISON AMONG CNNE, EPNET [53], EENCL [31], AND THREE OTHER ENSEMBLE ALGORITHMS [40] IN TERMS OF THE AVERAGE TER FOR THE DIABETES PROBLEM. THE RESULTS OF CNNE AND EPNET WERE AVERAGED OVER 30 RUNS, WHILE THEY WERE OBTAINED BY TWELVEFOLD AND FIVE STANDARD TENFOLD CROSS-VALIDATIONS FOR OTHER ALGORITHMS

	CNNE	EPNet	Bagging	Arc-boosting	Ada-boosting	EENCL
TER	0.198	0.224	0.228	0.244	0.233	0.221

TABLE XIII
COMPARISON AMONG CNNE, ACASPER [38], AND THREE OTHER ENSEMBLE ALGORITHMS [40] IN TERMS OF THE AVERAGE TER FOR THE GLASS PROBLEM. THE RESULTS OF CNNE AND ACASPER WERE AVERAGED OVER 30 AND 50 RUNS, RESPECTIVELY. THE RESULTS FROM OTHER ALGORITHMS WERE AVERAGED OVER FIVE STANDARD TENFOLD CROSS-VALIDATIONS

	CNNE	acasper	Bagging	Arc-boosting	Ada-boosting
TER	0.2680	0.3068	0.331	0.320	0.311

TABLE XIV
COMPARISON AMONG CNNE, EPNET [53] AND THREE OTHER ENSEMBLE ALGORITHMS [40] IN TERMS OF THE AVERAGE TER FOR THE HEART DISEASE PROBLEM. THE RESULTS OF CNNE AND EPNET WERE AVERAGED OVER 30 RUNS. THE RESULTS FROM OTHER ALGORITHMS WERE AVERAGED OVER FIVE STANDARD TENFOLD CROSS-VALIDATIONS

	CNNE	EPNet	Bagging	Arc-boosting	Ada-boosting
TER	0.1340	0.167	0.170	0.207	0.211

TABLE XV
COMPARISON AMONG CNNE AND THREE OTHER ENSEMBLE ALGORITHMS [40] IN TERMS OF THE AVERAGE TER ON THE LETTER RECOGNITION PROBLEM. THE RESULTS OF CNNE WERE OBTAINED BY AVERAGED OVER 30 RUNS, WHILE THEY WERE OBTAINED BY FIVE STANDARD TENFOLD CROSS-VALIDATIONS FOR BAGGING, ARC-BOOSTING, AND ADA-BOOSTING

	CNNE	Bagging	Arc-boosting	Ada-boosting
TER	0.062	0.105	0.057	0.046

Breast Cancer Problem: Table XI compares CNNEs results with those produced by EPNet [53] and three other ensemble algorithms tested by Opitz and Maclin [40]. CNNE, again, achieved the best performance although EPNet was a close second. One reason for the similar performance between CNNE and EPNet is that the cancer problem is a relatively easy problem. So improvement that can be made further by CNNE would be limited. However, CNNE was significantly faster than EPNet.

Diabetes Problem: Due to a relatively small dataset and high noise level, the diabetes problem is one of the most challenging problems in our experiments [53]. CNNE has outperformed all other algorithms significantly by a large margin. Table XII summarizes the results.

Glass Problem: The average results of CNNE, acasper [38] and three other ensemble algorithms tested by Opitz and Maclin [40] are given in Table XIII. Similar to the previous cases, CNNE outperformed all other algorithms significantly by a large margin.

Heart Disease Problem: Table XIV shows results from CNNE, EPNet [53] and three other ensemble algorithms tested by Opitz and Maclin [40]. It is clear from the table that CNNE performed the best among all algorithms.

Letter Recognition Problem: Table XV compares CNNEs results with three other ensemble algorithms tested by Opitz and Maclin [40]. In terms of average results, the three algorithms, i.e., bagging, arc-boosting, and ada-boosting, achieved testing error rates of 0.105, 0.057, and 0.046, respectively. Although the average performance of CNNE is better than bagging, ada-boosting performed significantly better than CNNE. It is well known that boosting may give significance improvement of performance when there is a reasonably large amount of data [13]. The results we obtained here appear to support this.

Soybean Problem: An important aspect of this problem is that most attributes have a significant number of missing values [42]. Thus, different selections of the testing set may significantly change the experimental results. This view seems to support the experimental results by Prechelt [42] who tested manually designed NNs by using three different testing sets. The average error rates of the manually designed NNs for the three testing sets were 0.088, 0.047, and 0.076. Opitz and Maclin [40] obtained the average testing error rate of 0.069, 0.067, and 0.063 for bagging, arc-boosting, and ada-boosting, respectively. CNNE achieved an average testing error rate of 0.076. Due to the significant number of missing values in the dataset, the ten-fold cross-validation used by bagging, arc-boosting, and ada-boosting would be able to average out some variations caused by different data splitting methods. Table XVI summarizes the results.

In order to gauge the performance of CNNE under cross-validation, additional experiments were carried out using tenfold cross-validation for CNNE, a setup that is the same as Optiz and Maclin's [40]. For each tenfold cross-validation, we first partitioned the original dataset into ten equal-sized sets, then each

TABLE XVI
COMPARISON AMONG CNNE, A HAND-DESIGNED NN [42] AND THREE OTHER ENSEMBLE ALGORITHMS [40] IN TERMS OF THE AVERAGE TER FOR THE SOYBEAN PROBLEM. THE RESULTS OF CNNE AND THE HAND-DESIGNED NN WERE AVERAGED OVER 30 RUNS, WHILE THEY WERE AVERAGED OVER FIVE STANDARD TENFOLD CROSS-VALIDATIONS FOR OTHER ALGORITHMS

	CNNE	Hand-designed NN			Bagging	Arc-boosting	Ada-boosting
		Testing Set 1	Testing Set 2	Testing Set 3			
TER	0.076	0.088	0.047	0.076	0.069	0.067	0.063

TABLE XVII
COMPARISON AMONG CNNE AND THREE OTHER ENSEMBLE ALGORITHMS [40] BASED ON THE AVERAGE TESTING ERROR RATE OVER FIVE RUNS. TENFOLD CROSS-VALIDATION WAS USED BY ALL ALGORITHMS

	CNNE	Bagging	Arc-boosting	Ada-boosting
Australian credit card	0.083	0.138	0.158	0.157
Cancer	0.011	0.034	0.038	0.040
Diabetes	0.178	0.228	0.244	0.233
Glass	0.246	0.331	0.320	0.311
Heart disease	0.112	0.170	0.207	0.211
Letter recognition	0.041	0.105	0.057	0.046
Soybean	0.060	0.069	0.067	0.063

set was in turn used as the test set while the remaining nine sets were used for training individual NNs in an ensemble like [40].

Table XVII compares the average results of CNNE with those given in [40] over five runs. It is clear that CNNE outperformed bagging, arc-boosting and ada-boosting employed by Optiz and Maclin [40] for all problem cases, including the letter recognition and soybean problems. These results appear to indicate that the experimental results using cross-validation tend to be an optimistic estimation of ensemble's generalization ability.

Based on the above comparisons, it is clear that CNNE performed better than other algorithms in most cases. Although such comparisons may not be entirely fair due to different experimental setups, we have tried our best to make our experimental setup as close to the previous ones as possible. Because of the diverse range of experimental setups used in previous studies, it is difficult (and probably unnecessary) to do an exhaustive comparison with all other work under different experimental setups. This is outside the scope of this paper.

5) Discussions: This section briefly explains why the performance of CNNE is better than other ensemble algorithms, e.g., CELS [31], EENCL [33], bagging, arc-boosting, and ada-boosting [40] for most classification problems we tested. There are three major differences that might contribute to better performance by CNNE in comparison with other ensemble algorithms.

The first reason is that CNNE emphasizes both accuracy and diversity among individual NNs in an ensemble, while CELS, EENCL, bagging, arc-boosting, and ada-boosting emphasize primarily diversity. The significance of accuracy can easily be understood by comparing the performance of CELS, EENCL, bagging, arc-boosting, and ada-boosting with that of acasper [38] and EPNet [53], which emphasizes accuracy but not diversity. Both acasper and EPNet could produce accurate single NNs. The performance of EPNet and acasper is better than or similar to that of EENCL, CELS, bagging, arc-boosting, and ada-boosting for most classification problems. For the diabetes problem, for example, the average testing error rate achieved by EPNet was 0.224, while EENCL, bagging, arc-boosting, and ada-boosting achieved the average testing error rates of 0.221, 0.228, 0.244, and 0.233, respectively. For the glass problem, the average testing error rate achieved by acasper was 0.306, while bagging, arc-boosting, and ada-boosting achieved the average testing error rates of 0.331, 0.320, and 0.311, respectively. CNNE achieved average testing error rates of 0.198 and 0.268, respectively, for the diabetes and glass problems (Tables IV and V).

The second reason is the effect of training procedure for individual NNs in an ensemble. In bagging, each individual NN in an ensemble was trained independently by a different training set. Each training set S' for an NN was constructed by forming a bootstrap replicate of the original training set S. The number of training epochs for each individual NN in an ensemble was the same. It was a user-specified fixed number. It is known that bagging can generate diverse networks only if the learning algorithm is "unstable," that is, only if small changes to the training set can cause large changes in NNs performance [8]. Breiman [5] noted that "the vital element (of bagging) is the instability of the prediction method. If perturbing the learning set can cause the significant changes in the prediction constructed, then bagging can improve accuracy." He also noted that "poor predictors can be transformed into worse one" by bagging [5]. This will never happen in CNNE due to its incremental nature.

Unlike bagging, boosting algorithms train individual NNs in an ensemble sequentially for a user-specified fixed number of training epochs. The algorithms maintain a set of weights over the original training set S and adjust these weights after each individual NN is trained by the learning algorithm. The adjustments increase the weight of examples that were misclassified by previously trained individual NN(s), and decrease the weight of examples that were correctly classified. The adjustment of weights leads to the construction of different training sets for different individual NNs in an ensemble. The only difference between arc-boosting and ada-boosting is the method used for calculating weights. Although boosting requires less instability than bagging, its effect is erratic [45]. It can overfit when training examples contain noise [40]. In addition, one may run out of examples to construct training sets for individual NNs in an ensemble when S contains a smaller number of examples.

In contrast, CNNE trains individual networks in an ensemble incrementally based on negative correlation learning [32]. Negative correlation learning encourages different individual NNs in an ensemble to learn different aspects of the training data. Although CNNE, CELS, and EENCL all use negative correlation learning for training individual NNs in an ensemble, the training strategy used in CNNE is different from that in EENCL and CELS. CNNE trains individual NNs incrementally for a different number of epochs, which is automatically determined, while EENCL and CELS simultaneously train individual NNs for a user specified and same number of training epochs. The advantage of incremental training and different training epochs for individual NNs used in CNNE is that they implement a form of early stopping [38], which has proven to be a useful method for obtaining good generalization [12], [37]. It has been shown that incremental training is better than simultaneous training both in terms of accuracy and diversity among individual NNs in an ensemble (Table IX).

The third is the effect of ensemble architectures. CNNE determines the number of individual NNs and their hidden nodes automatically. In contrast, CELS, bagging, arc-boosting, and ada-boosting all use user specified fixed ensemble architectures. According to [45], the performance of bagging and ada-boosting may vary significantly due to the variation of the number of individual NNs in an ensemble. Although EENCL determines the number of individual NNs in an ensemble automatically, it still determines the number of hidden nodes in individual NNs manually.

B. Mackey–Glass Chaotic Time Series Prediction Problem

CNNE performs well not only on classification problems as shown in the previous section, but also on regression and time series problems as demonstrated in this section. The aim of this section is to illustrate the applicability of CNNE to time-series problems. We apply CNNE to the Mackey–Glass time series prediction problem, which has been used in many studies in the NN community. This problem is different from previous classification problems in that its output is continuous.

The following delay-differential equation is used to generate Mackey–Glass time series for our study:

$$\dot{x}(t) = \beta x(t) + \frac{\alpha x(t-\tau)}{1 + x^{10}(t-\tau)} \tag{15}$$

where $\alpha = 0.2$, $\beta = -0.1$, and $\tau = 17$ [11], [34]. According to Martinetz *et al.* [35], $x(t)$ is quasi-periodic and chaotic with a fractal attractor dimension 2.1 for the above mentioned parameters.

1) Experimental Setup: The input to NNs consists of four data points, $x(t)$, $x(t-6)$, $x(t-12)$, and $x(t-18)$. The output is $x(t+6)$. To make multiple step prediction (e.g., $\Delta t = 90$) during testing, iterative prediction of $x(t+6), x(t+12), \ldots, x(t+90)$ will be made. During training, the true value of $x(t+6)$ was used as the target value.

For the following experiments, the fourth-order Runge–Kutta method with initial condition $x(0) = 1.2$, $(x - \tau) = 0$ for $0 \le t < \tau$ and the time step 1 was applied to (15) for producing the Mackey–Glass time series data. The training set consisted of points 118 to 617 (i.e., 500 training examples). The following 500 points, starting from 618, were used as the testing data. The value of training and testing data were rescaled between 0.1 and 0.9. The above mentioned experimental setup was chosen in order to make comparison possible and fair with other work.

According to [11] and [35], the normalized root mean square (rms) error E was used to evaluate the performance of ensembles, which is determined by the rms value of the absolute prediction error for $\Delta t = 6$, divided by the standard deviation of $x(t)$

$$E = \frac{\langle [x_p(t,\Delta t) - x(t+\Delta t)]^2 \rangle^{1/2}}{\langle (x - \langle x \rangle)^2 \rangle^{1/2}} \tag{16}$$

where $x_p(t,\Delta t)$ is the prediction of $x(t+\Delta t)$ from the current state x and $\langle x \rangle$ represents the expectation of x. As indicated in [11], "If $E = 0$, the predictions are perfect; $E = 1$ indicates the performance is no better than a constant predictor $x_p(t,\Delta t) = \langle x \rangle$."

TABLE XVIII
AVERAGE RESULTS PRODUCED BY CNNE OVER 30 RUNS FOR THE MACKEY–GLASS TIME SERIES PREDICTION PROBLEM

	Mean	SD	Min	Max
Number of networks	8.3	1.8	6.0	11.0
Number of hidden nodes	7.9	1.5	7.0	12.0
Error on test set ($\Delta = 6$)	0.009	0.0004	0.007	0.01
Error on test set ($\Delta = 84$)	0.028	0.0016	0.019	0.035
Error on test set ($\Delta = 90$)	0.0341	0.0021	0.0291	0.388

TABLE XIX
COMPARISON AMONG CNNE, CELS [33], AND EPNET [53] OVER 30 RUNS FOR THE MACKEY–GLASS TIME SERIES PREDICTION PROBLEM

Method	Testing RMS	
	$\Delta t = 6$	$\Delta t = 84$
CNNE	0.009	0.028
CELS	0.01	0.03
EPNet	0.02	0.06

2) Experimental Results and Comparison: Table XVIII shows the average results over 30 independent runs of CNNE. Each run was performed using a different set of random initial weights for individual NNs in the ensemble. Table XIX compares CNNEs results with those produced by EPNet [53], CELS [31], and EENCL [33]. It is clear from the table that CNNE outperformed all other algorithms for both $\Delta t = 6$ and $\Delta t = 84$. It is worth pointing out that CELS's results were obtained over 25 runs only [33], while EPNet [53] used a large amount of computational time in evolving the NN.

For a large time span with $\Delta t = 90$, CNNEs results were also better than those produced by Martinetz *et al.* [35]. For the same training set size of 500 data points, the *average* prediction error over 30 runs by CNNE was 0.034, which was smaller than the *smallest* prediction error (0.06) in [35].

V. CONCLUSION

Ensembles have been introduced to the NN community for nearly a decade. However, most ensemble training algorithms can only adjust weights in an ensemble. Few algorithms exist that design automatically an ensemble architecture, e.g., the number of NNs in an ensemble and the architecture of individual NNs. Ensemble design still has to rely on either a tedious trial-and-error process or an experienced human expert with rich prior knowledge about the problem to be solved. This paper proposes a new constructive algorithm to design as well as train NN ensembles. Neither the number of NNs in an ensemble nor the architecture of individual NNs need to be predefined and fixed. They are determined automatically in the learning process.

Our new algorithm, i.e., CNNE, adopts negative correlation learning [32], [33] to promote and maintain diversity among individual NNs in an ensemble. CNNE uses the constructive approach to grow individual NNs and an ensemble incrementally until the ensemble performance reaches a satisfactory level. Generalization is encouraged through the use of a validation set and different numbers of epochs for training different NNs. The criteria for growing NNs and the ensemble are based on an NNs contribution to reducing the ensemble's overall error, rather than in reducing its own error.

Extensive experiments have been carried out in this paper to evaluate how well CNNE performed on different problems in comparison with other ensemble and nonensemble algorithms. In almost all cases, CNNE outperformed the others. Although we compared the algorithms based on the averages only, the results have showed clearly the advantages of CNNE. For example, CNNEs average error rate for the Mackey–Glass time series prediction problem was even smaller than the smallest error rate produced by the "neural gas" algorithm (see Section IV-B2).

Although CNNE has performed very well for almost all problems we tested, our experimental study appeared to have revealed a weakness of CNNE in dealing with the letter recognition problem. Our initial speculation was that boosting algorithms were able to outperform CNNE because of their ability in exploiting a large training dataset. Boosting worked best when there was a large amount of training data. However, when we used the same cross-validation method as that used for boosting and bagging [40], the results from CNNE were actually better than those from boosting and bagging algorithms (Table XVII). It would be interesting in the future to analyze CNNE further and identify its strength and weakness. Potential hybridization between CNNE and boosting algorithms would also be an interesting future research topic.

Because the focus of this paper is on the presentation of the fundamental ideas and technical details of CNNE, the detailed comparison with other algorithms using rigorous statistical methods is left as the future work. Direct comparison with other algorithms using statistical tests is impractical at present, because some algorithms, such as boosting [40], etc., use cross-validation, while other do not, such as EENCL [31]. It is impossible to compare them fairly unless we reimplement all the algorithms under the same experimental setup.

Acknowledgment

The authors are grateful to the anonymous reviewers for their constructive comments, which helped to improve the clarity of this paper greatly.

References

[1] T. Ash, "Dynamic node creation in backpropagation networks," *Connection Sci.*, vol. 1, pp. 365–375, 1989.
[2] E. Bauer and R. Kohavi, "An empirical comparison of voting classification algorithms: Bagging, boosting, and variants," *Machine Learning*, vol. 36, pp. 105–139, 1999.
[3] J. C. Bioch, O.V.D. Meer, and R. Potharst, "Classification using Bayesian neural nets," in *Proc. IEEE Int. Conf. Neural Networks*, vol. 3, 1996, pp. 1488–1493.
[4] D. Bollé, D. R. C. Dominguez, and S. Amari, "Mutual information of sparsely coded associative memory with self-control and tenary neurons," *Neural Networks*, vol. 13, pp. 452–462, 2000.
[5] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, pp. 123–140, 1996.
[6] ——, "Stacked regressions," *Machine Learning*, vol. 24, pp. 49–64, 1996.
[7] ——, "Bias, Variance, and Arcing Classifier," Univ. California-Berkeley, Berkeley, CA, Tech. Rep. 460, 1996.
[8] T. G. Dietterich, "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization," *Machine Learning*, vol. 40, no. 2, pp. 139–157, 2000.
[9] H. Drucker, C. Cortes, L. D. Jackel, Y. LeCun, and V. Vapnik, "Boosting and other ensemble methods," *Neural Comput.*, vol. 6, pp. 1289–1301, 1994.
[10] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing System 2*, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1990, pp. 524–532.
[11] J. D. Farmer and J. J. Sidorowich, "Predicting the Mackey–Glass time series," *Phy. Rev. Lett.*, vol. 59, pp. 845–847, 1987.
[12] W. Finnoff, F. Hergent, and H. G. Zimmermann, "Improving model selection by nonconvergent methods," *Neural Networks*, vol. 6, pp. 771–783, 1993.
[13] Y. Freund and R. E. Schapire, "A decision theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, pp. 119–139, 1997.
[14] ——, "Experiments with a new boosting algorithm," in *Proc. 13th Int. Conf. Machine Learning*, 1996, pp. 148–146.
[15] P. Granitto, H. Navone, P. Verdes, and H. Ceccatto, "Late-stopping method for optimal aggregation of neural networks," *Int. J. Neural Syst.*, vol. 11, no. 3, pp. 305–310, 2001.
[16] P. Granitto, H. Navone, P. Verdes, and H. Ceccato, "A stepwise algorithm for construction of neural network ensembles," presented at the *VII Argentine Congr. Computer Science*, Calafate, Argentina, 2001.
[17] L. K. Hansen and P. Salamon, "Neural network ensembles," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 12, pp. 993–1001, Oct. 1990.
[18] S. Hashem, "Optimal linear combinations of neural networks," *Neural Networks*, vol. 10, no. 4, pp. 599–614, 1997.
[19] S. Hashem and B. Schmeiser, "Improving model accuracy using optimal linear combinations of trained neural networks," *IEEE Trans. Neural Networks*, vol. 6, pp. 792–794, July 1995.
[20] Y. Hirose, K. Yamashita, and S. Hijiya, "Backpropagation algorithm which varies the number of hidden units," *Neural Networks*, vol. 4, pp. 61–66, 1991.
[21] 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.
[22] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the EM algorithm," *Neural Comput.*, vol. 6, pp. 181–214, 1994.
[23] K. M. Ali, "Learning probabilistic relational concept descriptions," Ph.D. dissertation, Univ. California, Irvine, 1996.
[24] K. M. Ali and M. J. Pazzani, "Error reduction through learning multiple descriptions," *Machine Learning*, vol. 24, pp. 173–202, 1996.
[25] A. Khotanzad and C. Chung, "Hand written digit recognition using BKS combination of neural network classifiers," in *Proc. IEEE Southwest Symp. Image Anal. Interpretation*, 1994, pp. 94–99.
[26] A. Krogh and J. Vedelsby, "Neural network ensembles, cross validation, and active learning," in *Advances in Neural Information Processing Systems 7*, G. Tesauro, D. Touretzky, and T. Leen, Eds. Cambridge, MA: MIT Press, 1995, pp. 231–238.
[27] A. Krogh and P. Sollich, "Statistical mechanics of ensemble learning," *Phys. Rev. E*, vol. 55, pp. 811–825, 1997.
[28] T. Y. Kwok and D. Y. Yeung, "Constructive algorithms for structure learning in feedforward neural networks for regression problems," *IEEE Trans. Neural Networks*, vol. 8, pp. 630–645, May 1997.

[29] ——, “Objective functions for training new hidden units in constructive neural networks,” *IEEE Trans. Neural Networks*, vol. 8, pp. 1131–1148, Sept. 1997.

[30] M. Lehtokangas, “Modeling with constructive backpropagation,” *Neural Networks*, vol. 12, pp. 707–716, 1999.

[31] Y. Liu, X. Yao, and T. Higuchi, “Evolutionary ensembles with negative correlation learning,” *IEEE Trans. Evol. Comput.*, vol. 4, pp. 380–387, Sept. 2000.

[32] Y. Liu and X. Yao, “Ensemble learning via negative correlation,” *Neural Networks*, vol. 12, pp. 1399–1404, 1999.

[33] ——, “Simultaneous training of negatively correlated neural networks in an ensemble,” *IEEE Trans. Syst., Man, Cybern. B*, vol. 29, pp. 716–725, June 1999.

[34] M. C. Mackey and L. Glass, “Oscillation and chaos in physiological control systems,” *Science*, vol. 197, pp. 287–289, 1977.

[35] T. Martinetz, S. Berkovich, and K. Schulten, “’Neural-gas’ network for vector quantization and its application to time-series prediction,” *IEEE Trans. Neural Networks*, vol. 4, pp. 558–569, July 1993.

[36] M. M. Islam and K. Murase, “A new algorithm to design compact two-hidden-layer artificial neural networks,” *Neural Networks*, vol. 14, pp. 1265–1278, 2001.

[37] M. M. Islam, M. Shahjahan, and K. Murase, “Exploring constructive algorithms with stopping criteria to produce individual neural networks in the ensemble,” in *Proc. IEEE Syst., Man, Cybern.*, vol. 31, Oct. 2001, pp. 1526–1531.

[38] N. K. Treadgold and T. D. Gedeon, “Exploring constructive cascade networks,” *IEEE Trans. Neural Networks*, vol. 10, pp. 1335–1350, Nov. 1999.

[39] S. V. Odri, D. P. Petrovacki, and G. A. Krstonosic, “Evolutional development of a multilevel neural network,” *Neural Networks*, vol. 6, pp. 583–595, 1993.

[40] D. W. Opitz and R. Maclin, “Popular ensemble methods: An empirical study,” *J. Artificial Intell. Res.*, vol. 11, pp. 169–198, 1999.

[41] D. W. Opitz and J. W. Shavlik, “Actively searching for an effective neural-network ensemble,” *Connection Sci.*, vol. 8, no. 3/4, pp. 337–353, 1996.

[42] L. Prechelt, “Proben1 – A Set of Benchmarks and Benchmarking Rules for Neural Network Training Algorithms,” Univ. Karlsruhe, Karlsruhe, Germany, 1994.

[43] ——, “A quantitative study of experimental evaluation of neural network learning algorithms,” *Neural Network*, vol. 9, pp. 457–462, 1996.

[44] ——, “Some notes on neural learning algorithm benchmarking,” *Neurocomput.*, vol. 9, pp. 343–347, 1995.

[45] J. R. Quinlan, “Bagging, boosting, and C4.5,” in *Proc. 13th Nat. Conf. Artificial Intelligence 8th Innovative Applications of Artificial Intelligence Conf.*, Menlo Park, Aug. 4–8, 1996, pp. 725–730.

[46] B. E. Rosen, “Ensemble learning using decorrelated neural networks,” *Connection Sci.*, vol. 8, pp. 373–383, 1996.

[47] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation,” in *Parallel Distributed Processing*, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: MIT Press, 1986, vol. I, pp. 318–362.

[48] A. J. C. Sharkey, “On combining artificial neural nets,” *Connection Sci.*, vol. 8, no. 3/4, pp. 299–314, 1996.

[49] A. J. C. Sharkey and N. E. Sharkey, “Combining diverse neural nets,” *Knowledge Eng. Rev.*, vol. 12, no. 3, pp. 1–17, 1997.

[50] R. Setiono and L. C. K. Hui, “Use of quasi-Newton method in a feed forward neural network construction algorithm,” *IEEE Trans. Neural Networks*, vol. 6, pp. 273–277, Mar. 1995.

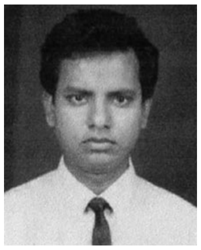
[51] K. Tumer and J. Ghosh, “Error correlation and error reduction in ensemble classifiers,” *Connection Sci.*, vol. 8, pp. 385–404, 1996.

[52] ——, “Linear and order statistics combiners for pattern classification,” in *Combining Artificial Neural Nets*, A. Sharkey, Ed. New York: Springer-Verlag, 1999, pp. 127–162.

[53] X. Yao and Y. Liu, “A new evolutionary system for evolving artificial neural networks,” *IEEE Trans. Neural Networks*, vol. 8, pp. 694–713, May 1997.

[54] ——, “Making use of population information in evolutionary artificial neural networks,” *IEEE Trans. Syst., Man, Cybern. B*, vol. 28, pp. 417–425, June 1998.

[55] Z. Zhou, J. Wu, Y. Jiang, and S. Chen, “Genetic algorithm based selective neural network ensemble,” in *Proc. 17th Int. Joint Conf. Artificial Intelligence*, vol. 2, Seattle, WA, 2001, pp. 797–802.



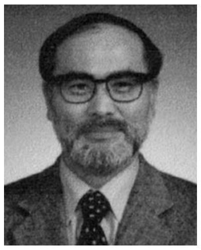
Md. Monirul Islam received the B.E. degree in electrical and electronic engineering from the Bangladesh Institute of Technology (BIT), Khulna, Bangladesh, in 1989, the M.E. degree in computer science and engineering from Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh, in 1996, and the Ph.D. degree in evolutionary robotics from Fukui University, Fukui, Japan, in 2002.

From 1989 to 2002, he was a Lecturer and Assistant Professor with the Department of Electrical and Electronic Engineering, BIT. In 2003, he moved to BUET as an Assitant Professor in the Department of Computer Science and Engineering. His research interests include evolutionary robotics, evolutionary computation, neural networks, and pattern recognition.

Xin Yao (S’91–SM’96–F’03) received the B.Sc. degree from the University of Science and Technology of China (USTC), Hefei, China, in 1982, the M.Sc. degree from the North China Institute of Computing Technology, Beijing, in 1985, and the Ph.D. degree from USTC in 1990.

From 1985 to 1990, he was an Associate Lecturer and Lecturer at USTC. He took up a postdoctoral fellowship in the Computer Sciences Laboratory, Australian National University (ANU), Canberra, in 1990, and continued his work on simulated annealing and evolutionary algorithms. He joined the Knowledge-Based Systems Group, CSIRO Division of Building, Construction and Engineering, Melbourne, Australia, in 1991, working primarily on an industrial project on automatic inspection of sewage pipes. He returned to Canberra in 1992, to take up a lectureship in the School of Computer Science, University College, the University of New South Wales (UNSW), the Australian Defence Force Academy (ADFA), where he was later promoted to a Senior Lecturer and Associate Professor. In 1999, he moved to the University of Birmingham, Birmingham, U.K., as a Professor of Computer Science. Currently, he is the Director of the Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA), which was set up through a substantial grant from the Advantage West Midlands. His current work include evolutionary artificial neural networks, automatic modularization of machine learning systems, evolutionary optimization, constraint handling techniques, computational time complexity of evolutionary algorithms, iterated prisoner’s dilemma, and data mining.

Dr. Yao is Editor-in-Chief of IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and Associate Editor of several other journals. He chairs the IEEE NNS Technical Committee on Evolutionary Computation and has chaired/cochaired more than 25 international conferences and workshops. He has given more than 20 invited keynote/plenary speeches at conferences and workshops world-wide. He won the 2001 IEEE Donald G. Fink Prize Paper Award for his work on evolutionary artificial neural networks. His Ph.D. work on simulated annealing and evolutionary algorithms was awarded the President’s Award for Outstanding Thesis by the Chinese Academy of Sciences.



Kazuyuki Murase received the M.E. degree in electrical engineering from Nagoya University, Nagoya, Japan, in 1978, and the Ph.D. degree in biomedical engineering from Iowa State University, Ames, in 1983.

He was a Research Associate at the Department of Information Science, Toyohashi University of Technology, Toyohashi, Japan, from 1984, and then moved to Fukui University, Fukui, Japan, in 1988 as an Associate Professor at the Department of Information Science and became a Professor in 1993. He served as the Chairman of the newly established Department of Human and Artificial Intelligence Systems (HART) of Fukui University in 1999. He has been working on neuroscience of sensory systems, self-organizing neural networks, and bio-robotics.