

A New Constructive Algorithm for Architectural and Functional Adaptation of Artificial Neural Networks

Md. Monirul Islam, Md. Abdus Sattar, Md. Faijul Amin, Xin Yao, *Fellow, IEEE*, and Kazuyuki Murase

Abstract—The generalization ability of artificial neural networks (ANNs) is greatly dependent on their architectures. Constructive algorithms provide an attractive automatic way of determining a near-optimal ANN architecture for a given problem. Several such algorithms have been proposed in the literature and shown their effectiveness. This paper presents a new constructive algorithm (NCA) in automatically determining ANN architectures. Unlike most previous studies on determining ANN architectures, NCA puts emphasis on architectural adaptation and functional adaptation in its architecture determination process. It uses a constructive approach to determine the number of hidden layers in an ANN and of neurons in each hidden layer. To achieve functional adaptation, NCA trains hidden neurons in the ANN by using different training sets that were created by employing a similar concept used in the boosting algorithm. The purpose of using different training sets is to encourage hidden neurons to learn different parts or aspects of the training data so that the ANN can learn the whole training data in a better way. In this paper, the convergence and computational issues of NCA are analytically studied. The computational complexity of NCA is found to be $O(W \times P_t \times \tau)$, where W is the number of weights in the ANN, P_t is the number of training examples, and τ is the number of training epochs. This complexity has the same order as what the backpropagation learning algorithm requires for training a fixed ANN architecture. A set of eight classification and two approximation benchmark problems was used to evaluate the performance of NCA. The experimental results show that NCA can produce ANN architectures with fewer hidden neurons and better generalization ability compared to existing constructive and nonconstructive algorithms.

Index Terms—Architectural adaptation, artificial neural networks (ANNs), constructive approach, functional adaptation, generalization ability.

Manuscript received December 10, 2008; revised March 27, 2009. First published June 5, 2009; current version published November 18, 2009. The work of M. Islam was supported in part by a fellowship grant from the Japanese Society for the Promotion of Science. This paper was recommended by Associate Editor E. Santos, Jr.

Md. M. Islam is with the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka 1000, Bangladesh, and also with the Department of Human and Artificial Intelligence Systems, Graduate School of Engineering, University of Fukui, Fukui 910-8507, Japan (e-mail: monirul@synapse.his.fukui-u.ac.jp).

Md. A. Sattar is with the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka 1000, Bangladesh.

Md. F. Amin is with the Department of Human and Artificial Intelligence Systems, University of Fukui, Fukui 910-8507, Japan.

X. Yao is with the Center of Excellence for Research in Computational Intelligence and Applications, School of Computer Science, University of Birmingham, B15 2TT Birmingham, U.K.

K. Murase is with the Department of Human and Artificial Intelligence Systems, Graduate School of Engineering, University of Fukui, Fukui 910-8507, Japan, and also with the Research and Education Program for Life Science, University of Fukui, Fukui 910-8507, Japan.

Digital Object Identifier 10.1109/TSMCB.2009.2021849

I. INTRODUCTION

GENERALIZATION ability of artificial neural networks (ANNs) is greatly dependent on their architectures. There have been many attempts in automatically determining ANN architectures, such as various constructive, pruning, constructive-pruning, and evolutionary algorithms [1]–[7]. Roughly speaking, a constructive algorithm starts with a minimal ANN architecture and adds new layers, nodes, and connections, if necessary, during training, whereas a pruning algorithm does the opposite, i.e., it deletes unnecessary layers, nodes, and connections from a large ANN architecture. A constructive-pruning algorithm is a hybrid approach that executes, first, a constructive phase and, then, a pruning phase. Although an evolutionary algorithm that was used in determining ANN architectures also executes both the constructive and pruning phases, it does not follow any predefined sequence in executing the constructive or pruning phase.

We can formulate architecture determination for an ANN as an optimization problem. The solution of this optimization problem is the “complete” topological information of the ANN, i.e., the number of hidden layers in the ANN and of neurons in each hidden layer. Although single-hidden-layered ANNs can solve any problem with arbitrary accuracy [8], multiple-hidden-layered ANNs have shown to perform better for some problems [9]–[14]. An empirical study, however, showed the superiority of training fixed single-hidden-layered ANNs to fixed two-hidden-layered ANNs [15]. Tamura and Tateishi [16] later analytically proved the superiority of two-hidden-layered ANNs to single-hidden-layered ANNs. It therefore stands to reason that the superiority of single- or multiple-hidden-layered ANNs is not conclusive; at the same time, it is not appropriate to train ANNs with a fixed number of hidden layers (or hidden neurons) and adapt either number during training.

This paper describes a new constructive algorithm (NCA) to determine complete topological information and some particular weight values of an ANN. The new approach combines architectural adaptation with functional adaptation in one scheme. It starts adaptation with a minimal ANN architecture and the whole training data. As adaptation progresses, NCA gradually adds hidden neurons or hidden layers to the ANN and excludes the examples of training data that are already learned. NCA’s emphasis on architectural adaptation and functional adaptation can improve the performance of an architecture determination process [17].

The proposed NCA is different from previous works on several aspects. First, NCA determines the number of hidden layers in an ANN and of hidden neurons in each hidden layer. Most existing algorithms determine the number of hidden layers or hidden neurons but not both (for example, see the review papers

[1], [18] and some recent works [17], [19]–[22]). The problem of using a fixed number of hidden layers or hidden neurons lies in the difficulty of selecting an appropriate number that is suitable for all problems. Although the tiling algorithm [23] determines both numbers, the algorithm can operate only on binary-valued attributes and does not consider functional adaptation. However, NCA operates on binary- or real-valued attributes and considers both functional and architectural adaptation.

Second, NCA trains each hidden neuron in an ANN with a different training set. When NCA adds a new neuron in a hidden layer, it creates a new training set based on the performance of the existing ANN architecture. The examples in the training set for which the existing architecture gets wrong and correct are “boosted” and “weakened,” respectively, in the new training set. Although this technique is adopted in the boosting algorithm [24] for training ensembles [25], [26], this paper is the first attempt, to the best of our knowledge, to use a similar concept for training a single ANN. The goal of this training strategy is to encourage a hidden neuron to focus on unsolved parts or aspects of the training data so that the ANN can learn the whole training data in a better way. This training strategy is completely different from most (perhaps all) existing strategies that train all hidden neurons of the ANN using the same training data.

Third, NCA uses a new criterion for adding a hidden layer to an existing ANN architecture. The new criterion utilizes both the training error and the hidden neurons’ functionality. The most common practice in adding a hidden layer is based only on the training error. One problem with this layer-addition criterion is that an algorithm may add a hidden layer, regardless if it is necessary or not, because the training error gradually decreases when the algorithm adds neurons one by one either in an existing hidden layer or in a new hidden layer. Thus, it would be very difficult to decide whether a neuron should be added in an existing hidden layer or in a new hidden layer. One criterion that adds a hidden layer when neurons in the existing hidden layer exhibit a similar functionality and the training error does not reduce much can be a simple solution of the aforementioned problem. This instance is the main reason for using hidden neurons’ functionality in the layer-addition criterion of NCA.

The rest of this paper is organized as follows. Section II discusses two important components of constructive algorithms and explains why we adopted these components in a different way to our NCA. Section III describes our proposed algorithm in detail. Section IV presents the results of our experimental study. Finally, Section V concludes this paper with a brief summary and few remarks.

II. HIDDEN NEURON ADDITION AND TRAINING IN CONSTRUCTIVE ALGORITHMS

Two important components of constructive algorithms are neuron addition and training. In this section, we will describe these components in detail.

A. Neuron Addition

Constructive algorithms generally add one or few hidden neurons in each addition step [27]. The neurons are added in the same hidden layer for single-hidden-layered ANNs or different hidden layers for multiple-hidden-layered ANNs. The addition of neurons in the single-hidden-layered ANNs is very

straightforward, because one can add one or few neurons when a neuron-addition criterion is satisfied. A large number of constructive algorithms have been proposed, which add neurons in single-hidden-layered ANNs (see the review paper [1] and recent works [5], [17]).

The addition of neurons in different hidden layers, however, is not straightforward. It is because one has to decide where a neuron will be added: in an existing hidden layer or in a new hidden layer. To tackle this issue, most existing algorithms add a predefined and fixed number of neurons in the first hidden layer, then add the same number of neurons in the second hidden layer, and so on (see, for example, [14], [28], and [29]). As aforementioned, this number is very crucial for the performance of ANNs, and restricting it to a small value limits the ability of a hidden layer to form complicated feature detectors [1]. In this paper, we propose a new layer-addition criterion (see Section III-D). The novelty of our criterion is that it can automatically add a hidden neuron to an existing hidden layer or a new hidden layer. Each hidden layer, therefore, may contain a different number of neurons, and one does not need to decide how many hidden neurons will be added in each hidden layer of an ANN.

B. Issues of Training

The second important issue of constructive algorithms is the way that an ANN is trained after adding hidden neurons. This issue is also important for pruning, constructive-pruning, and evolutionary algorithms. It is because, although ANNs with fixed architectures are trained only once, ANNs must be trained every time when their architectures are modified by any algorithm. Hence, the computational efficiency of training becomes an important issue. NCA is a constructive approach; thus, we briefly summarize here two major schemes that are widely used in constructive approaches.

One scheme is to train all weights of an ANN (e.g., [17] and [27]), and the other scheme is to train only the weights that are associated with a newly added hidden neuron, keeping all other weights unchanged (fixed) (e.g., [14], [28], and [29]). The former scheme is very simple and straightforward, because it trains all weights of the ANN after each addition step. The main disadvantage of this approach is the computational efficiency. In addition, this scheme suffers from the so-called *moving-target* problem, where each hidden node ‘sees’ a constantly moving environment [14]. The exact computational requirement depends on a particular learning method for training. For example, the Newton method requires $O(W^3)$ operations in each iteration to train an ANN with W weights, a quasi-Newton method requires $O(W^2)$ operations, and a simple gradient descent method requires $O(W)$ operations. Due to their computational inefficiency, small-sized ANNs have been trained in previous studies using this scheme [1]. The latter scheme [14] trains only the weights of a newly added neuron at a time while keeping the weights of all previously trained hidden neurons fixed. Although this scheme achieves computational efficiency, quickly converges, and avoids the moving-target problem [14], it may produce networks with a large number of hidden neurons. This situation will arise when a scheme fixes the weights of hidden neurons before properly optimizing them. To overcome this problem, NCA introduces a two-stage training scheme for hidden neurons of an ANN.

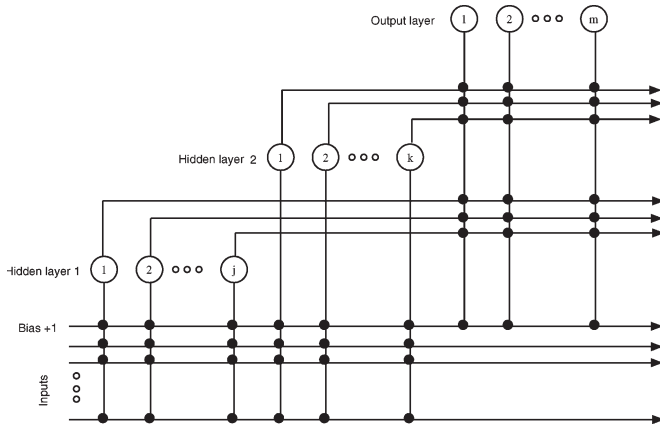


Fig. 1. An ANN architecture with two hidden layers.

In addition to computational efficiency, another important aspect of training is the data of a learning task. When constructive algorithms add a neuron, the objective is to solve the remaining unsolved parts of a given learning task. The proposed NCA tries to achieve this objective by employing AdaBoost [30], which is a variant of boosting algorithm [24], to create new training sets for hidden neurons of an ANN (see Section III). Although some previous work acknowledged (e.g., [27]) the importance of incorporating the boosting algorithm in the training process of the ANN, no techniques have been developed so far. The most common practice was to train each hidden neuron in the ANN using the same training set. Such an approach tends to produce hidden neurons with the nearly same functionality; therefore, this approach does not help in achieving functional adaptation in determining ANN architectures.

III. NCA

This section presents our proposed NCA in detail. In its current implementation, NCA is used to determine the complete topological information of feedforward ANN architectures with sigmoid hidden neurons. Each hidden layer of such architecture receives signals from all preceding layers (i.e., the input plus the preceding hidden layers), whereas the output layer receives signals from all hidden layers (see Fig. 1). The reason for using these particular traversal paths for signals is to facilitate a fair comparison with previous work. In fact, NCA has no constraint on the type of ANNs. The feedforward ANNs do not have to be strictly layered, fully connected between adjacent layers, or be of any other connectivity. They may also contain hidden neurons with different activation functions [31].

The major steps of NCA are summarized in Fig. 2, which are further explained as follows.

Step 1) Choose a minimal ANN architecture with three layers: 1) an input layer; 2) a hidden layer; and 3) an output layer. The number of neurons in the input and output layers is the same as the number of inputs and outputs of a given problem, respectively. Initially, the hidden layer contains one neuron. Randomly initialize all connection weights of the ANN within a small range. Label the hidden layer and its neuron with C and I , respectively.

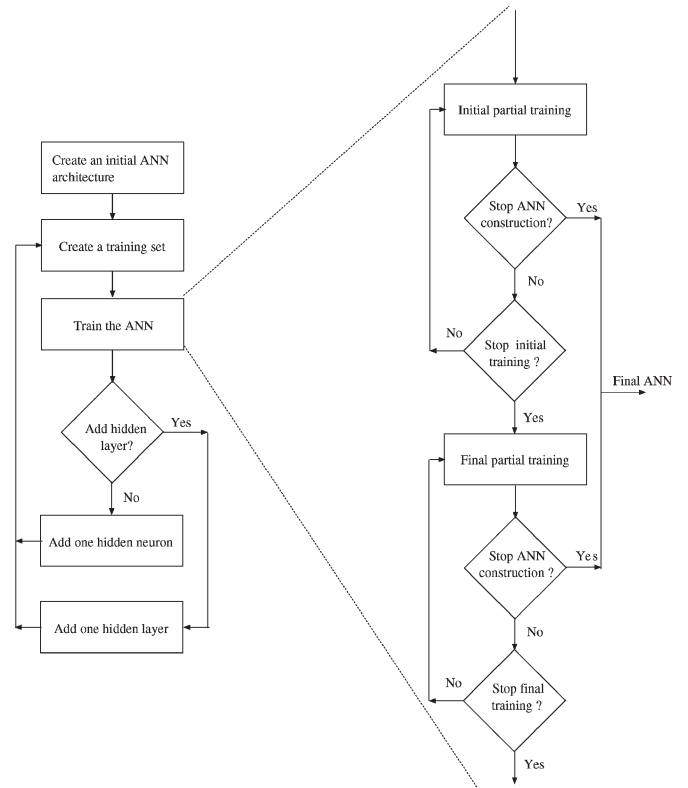


Fig. 2. Major steps of NCA.

- Step 2) Create a new training set for the I -labeled hidden neuron of the C -labeled hidden layer. Note that NCA does not create any training set for the first I -labeled hidden neuron of the first C -labeled hidden layer. The original training set is considered here as the new training set for this hidden neuron.
- Step 3) Train the I -labeled hidden neuron of the C -labeled hidden layer on the new training set using the backpropagation (BP) learning algorithm [32] for a certain number of training epochs. The number of training epochs τ is specified by the user. We call this training phase as the *initial partial training* for the I -labeled hidden neuron.
- Step 4) Check the termination criterion for stopping the ANN construction process. If this criterion is satisfied, go to Step 11). Otherwise, continue.
- Step 5) Compute the ANN error E on the training set. If this error reduces by a predefined amount ϵ_1 after training for τ epochs, go to Step 3). It is assumed that training is progressing well and further training is necessary. Otherwise, continue. According to [33], E is calculated as

$$E = 100 \frac{o_{\max} - o_{\min}}{NP_t} \sum_{p=1}^{P_t} \sum_{i=1}^m (Y_i(p) - Z_i(p))^2 \quad (1)$$

where o_{\max} and o_{\min} are the maximum and minimum values of the output coefficients in a problem representation, P_t is the number of examples in the training set, and m is the number of output neurons. The value for o_{\max} and o_{\min} could be same as the

maximum and the minimum target values, i.e., 1 and 0, for classification problems. The variables $Y_i(p)$ and $Z_i(p)$ represent the actual and desired outputs of the i th output neuron for a training example p . Prechelt [33] suggested (1) to make the error measure less dependent on the size of a training set and the number of output nodes. Hence, the mean squared error percentage was adopted.

- Step 6) Add a small amount of noise to all input and output connection weights of the I -labeled hidden neuron of the C -labeled hidden layer. Gaussian distribution with a mean of zero and a variance of one is used to add a small amount of noise. It is worth mentioning that NCA adds noise to the connection weights of any I -labeled hidden neuron only once. Train the I -labeled hidden neuron using BP [32] for τ epochs. We call this training phase as the *final partial training* for the I -labeled hidden neuron.
- Step 7) Check the termination criterion for stopping the ANN construction process. If this criterion is satisfied, go to Step 11). Otherwise, continue.
- Step 8) Compute E on the training set. If E is reduced by an amount ϵ_1 after training τ epochs, go to Step 6) for further training of the I -labeled hidden neuron. Otherwise, freeze (keep fixed) the input and output connection weights of the I -labeled hidden neuron, remove the label of the I -labeled hidden neuron, and continue.
- Step 9) Check the criterion for adding a new hidden layer. If the criterion is not satisfied, add a new neuron to the C -labeled hidden layer and go to Step 2). Label the new neuron with I and initialize its input and output connection weights with zero. Otherwise, remove the label of the C -labeled hidden layer and continue. The reason for initializing the weights with zero is to start further training from the previous error value.
- Step 10) Add a new hidden layer with one neuron above the existing hidden layer(s) of the ANN. Label the new hidden layer with C and its neuron with I . Initialize the input and output connection weights of the neuron with zero and go to Step 2).
- Step 11) The existing network architecture is the final ANN for the given problem.

The essence of NCA is the utilization of a constructive approach with the following four components:

- 1) creation of new training sets;
- 2) two-stage training;
- 3) termination criterion;
- 4) architecture adaptation.

Details about each of these components, along with convergence issues and computational complexity, are given in the following sections.

A. Creation of New Training Sets

The proposed NCA employs AdaBoost [30] to create a new training set. The AdaBoost algorithm is widely used to train an ensemble of ANNs or decision trees. This algorithm maintains a probability distribution D over an original training set T . We have a finite number of examples in T ; thus, D can be

considered a vector of dimension equal to the number of training examples. Each component of D denotes the probability that each example will be selected for a training set. Initially, the component of D is set at $1/P_t$, where P_t is the number of examples in T . The AdaBoost algorithm trains the first ANN in the ensemble using the original training set T . It then updates D based on the performance of the trained ANN, where the probability of correctly and incorrectly classified examples is decreased and increased, respectively. Finally, AdaBoost creates a new training set T' based on updated D by sampling P_t examples at random with replacement from T and trains the second ANN in the ensemble using T' . The whole process is repeated to train other ANNs in the ensemble.

NCA trains only one neuron at a time; thus, the strategy in AdaBoost for creating a new training set can easily be incorporated in NCA, because NCA adds hidden neurons in an ANN one by one as AdaBoost adds ANNs in an ensemble. Furthermore, NCA trains only one, i.e., a newly added hidden neuron of the ANN. This case is similar to training only one ANN of the ensemble by AdaBoost. The use of a different training set in NCA facilitates functional adaptation during adapting ANN architectures. This approach of functional adaptation is quite different from a recently proposed approach that uses different activation functions for different hidden neurons of the ANN [17]. Although different activation functions facilitate hidden neurons to differently perform, it is not sure whether these neurons focus on different unsolved parts of a training data, because it is the training data on which a network is trained that mostly determines the function it approximates.

B. Two-Stage Training

Our NCA uses a training scheme with two stages: 1) initial and 2) final. We use the BP learning algorithm [32] in both stages. In the initial training stage, NCA first initializes the input and output connection weights of a newly added neuron with zero. The only exception is the first hidden neuron in an ANN for which the weights are randomly initialized within a small range. The BP algorithm then trains the hidden neuron to modify its weights. When the training error does not reduce by a predefined amount, i.e., ϵ_1 , after training for τ epochs, NCA assumes that the weights are trapped into local optima. This assumption is reasonable, because BP is notorious for its slow convergence and convergence to local optima [7]. To alleviate these problems, NCA introduces the final training stage, where a small amount of Gaussian noise with a mean of zero and a variance of one is added to the weights of the initially trained hidden neuron. The BP learning algorithm is then reapplied to optimize those weights. Both theoretical and empirical studies have demonstrated that adding noise in the connection weights improves the convergence rate and generalization ability of ANNs [34].

Most existing constructive algorithms use a one-stage training scheme. They add a hidden neuron to an ANN when its training error does not reduce by some amount ϵ_1 after training the ANN for τ epochs. An unsatisfactory error reduction is assumed due to a small number of hidden neurons in the ANN, although it is not always true. Accordingly, existing constructive algorithms produce ANNs with more hidden neurons. Some algorithms (e.g., [5], [12], and [13]) therefore execute

a pruning phase after the addition phase to produce compact ANN architectures. There are only few algorithms (e.g., CCLA [14] and its variants [28]) that use a two-stage training. The problems of their two-stage training are the necessity of two cost functions (one for training and one for adding hidden neuron) and of training a pool of hidden neurons for adding only one hidden neuron.

We have now understood that the two-stage training scheme in NCA is advantageous, because it uses one cost function and trains only one hidden neuron for adding one hidden neuron. The purpose of two-stage training is to increase the capabilities of a hidden neuron by properly optimizing its weights. If a training scheme cannot produce hidden neurons with good capabilities, an ANN will require more hidden neurons to solve a problem. To produce ANNs with a smaller number of hidden neurons, NCA uses the two-stage training scheme. A hybrid training scheme that uses BP [32] in initial training and simulated annealing [35] in final training can also be used in our NCA. The investigation of the best training scheme is outside the scope of this paper and would be the topic of a separate paper.

C. Termination Criterion

Prechelt [36] describes a number of termination criteria to stop training for an ANN. In this paper, one of these criteria that utilize both training and validation errors is employed. To formally describe this criterion, let $E_{tr}(\tau)$ and $E_{va}(\tau)$ be the training and validation errors at training epoch τ , respectively, and $E_{opt}(\tau)$ be the lowest validation up to epoch τ . According to [36], the generalization loss at epoch τ , i.e., $GL(\tau)$, can be defined as

$$GL(\tau) = \left(\frac{E_{va}(\tau)}{E_{opt}(\tau)} - 1 \right). \quad (2)$$

A high generalization loss can be one obvious reason for stopping the training, because it directly indicates overfitting. However, if E_{tr} very rapidly reduces, it is desirable not to stop the training, because the generalization losses have a higher chance to be repaired [36]. The training progress $P_k(\tau)$ at training epoch τ of a training strip k can be used to measure how much the average training error of the strip is larger than the minimum training error during the strip. Hence, $P_k(\tau)$ can be defined as [36]

$$P_k(\tau) = \left(\frac{\sum_{\tau'=\tau-k+1}^{\tau} E_{tr}(\tau')}{k \times \min_{\tau'=\tau-k+1}^{\tau} E_{tr}(\tau')} - 1 \right) \quad (3)$$

where the strip length k is a sequence of epochs numbered $n + 1, n + 2, \dots, n + k$, and n is divisible by k . The value of k is generally set to 5 [36], which is used in this paper. Our NCA stops the training of an ANN when $GL(\tau) > P_k(\tau)$. This criterion is a bit complex than the one that stops the training when a validation error begins to increase. However, the essence of using both training and validation errors is that they can anticipate the behavior of the test data better [36].

D. Architecture Adaptation

When the two-stage training fails to improve the performance of an ANN, NCA modifies the ANN by adding hidden neurons

or hidden layers. The proposed NCA uses two simple criteria to decide when to add a hidden neuron and when to add a hidden layer. The following equations are used to describe the following criteria:

$$E(i) - E(i + \tau) \leq \epsilon_1, \quad i = \tau, 2\tau, 3\tau, \dots \quad (4)$$

$$d = \frac{1}{P_t} \sum_{p=1}^{P_t} |y_k(p) - y_{k+1}(p)| \leq \epsilon_2, \quad k = 1, 2, 3, \dots \quad (5)$$

where $E(i)$ and $E(i + \tau)$ are the ANN errors at epochs i and $(i + \tau)$, respectively. The error threshold ϵ_1 is a user-specified small positive number. The variables $y_k(p)$ and $y_{k+1}(p)$, respectively, are the outputs of two previously added hidden neurons h_k and h_{k+1} for the p th training example, whereas P_t is the number of examples in h_{k+1} 's training set. The output threshold ϵ_2 is also a user-specified small positive number.

The proposed NCA adds one neuron to a C -labeled existing hidden layer when the condition described by (4) is satisfied, but the condition described by (5) is not satisfied. On the contrary, most existing constructive algorithms use only (4) to add hidden neurons, i.e., they do not consider the functionality of hidden neurons. Our NCA adds a new hidden layer with one neuron above the existing hidden layer(s), provided that both the conditions described by (4) and (5) are satisfied. It is here assumed that adding more neurons in the same hidden layer is not beneficial, because some hidden neurons are likely to be redundant due to a similar functionality. Fig. 1 shows that each hidden layer receives signals from all preceding layers. Thus, two hidden neurons from two different hidden layers must exhibit different functionalities, because they receive different signals from the preceding layers (see Fig. 1). This instance is the main reason that NCA adds a new hidden layer when hidden neurons become functionally similar.

E. Convergence Issues

To discuss the convergence issue of NCA, we restrict ourselves to a function approximation task with one output. This restriction is applied without loss of generality and for simplicity. The output neuron of an ANN uses the identity activation function for the approximation problem. That is, the output neuron just produces a weighted sum of all its inputs. The function that was approximated by the ANN can therefore be expressed as

$$\hat{f}(x) = \sum_{j=1}^n \beta_j g_j(x) \quad (6)$$

where g_j represents the j th input to the output neuron, and n denotes the total number of input connections to the output neuron after the ANN construction has been completed. β_j is the weight of the j th input connection to the output neuron.

Each g_j in NCA is the output of a hidden neuron of some hidden layer. Let Γ be the set of all functions that can be implemented by hidden units. Thus, $g_j \in \Gamma$. The particular form of g_j depends on how the corresponding hidden neuron is connected. As shown in Section III, NCA incrementally adds each g_j , keeping all β_k and g_k , for $1 \leq k \leq j - 1$, fixed

(frozen). Hence, after the n th input connection is given to the output neuron, the function approximated by the ANN is

$$\begin{aligned} f_n(x) &= \beta_n g_n(x) + \sum_{j=1}^{n-1} \beta_j g_j(x) \\ &= \beta_n g_n(x) + f_{n-1}(x). \end{aligned} \quad (7)$$

Henceforth, we will drop the argument of the function, i.e., we will use f_n instead of $f_n(x)$. The norm in L^2 space will be denoted as $\|\cdot\|$. The convergence proof of a constructive algorithm given here is the most general one, which assumes that this algorithm in each incremental step can find a g_j according to the requirement of the proof. In [37], such a convergence proof is given. We briefly restate this proof here for subsequent explanation. Later, we will show how NCA selects each g_j to proceed and construct \hat{f} according to the requirement of convergence.

Proposition 1: For a fixed $g \in \Gamma(\|g\| \neq 0)$,¹ the expression $\|f - (f_{n-1} + \beta g)\|$ achieves its minimum if and only if

$$\beta = \beta^* = \frac{\langle e_{n-1}, g \rangle}{\|g\|^2} \quad (8)$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product, and $e_{n-1} = f - f_{n-1}$, with f being the target function that will be approximated by the ANN. Moreover, with β_n^* and β^* as defined in (8), $\|f - (f_{n-1} + \beta_n^* g_n)\| \leq \|f - (f_{n-1} + \beta^* g)\| \forall g \in \Gamma$, if and only if

$$\frac{\langle e_{n-1}, g_n \rangle^2}{\|g_n\|^2} \geq \frac{\langle e_{n-1}, g \rangle^2}{\|g\|^2} \quad \forall g \in \Gamma.$$

Proof: We see that

$$\begin{aligned} \Delta e(g) &= \|f - f_{n-1}\|^2 - \|f - (f_{n-1} + \beta g)\|^2 \\ &= \|e_{n-1}\|^2 - \|e_{n-1} - \beta g\|^2. \end{aligned} \quad (9)$$

Based on the triangle law for vector addition [37]

$$\|e_{n-1}\|^2 = \|\beta g\|^2 + \|e_{n-1} - \beta g\|^2 + 2\langle \beta g, (e_{n-1} - \beta g) \rangle.$$

Thus, according to (9)

$$\begin{aligned} \Delta e(g) &= \|\beta g\|^2 + 2\langle e_{n-1}, \beta g \rangle - 2\beta^2 \|g\|^2 \\ &= 2\beta \langle e_{n-1}, g \rangle - \beta^2 \|g\|^2 \\ &= \frac{\langle e_{n-1}, g \rangle^2}{\|g\|^2} - \left(\frac{\langle e_{n-1}, g \rangle}{\|g\|} - \beta \|g\| \right)^2. \end{aligned}$$

With a fixed g , $\Delta e(g)$ will be maximum if

$$\frac{\langle e_{n-1}, g \rangle}{\|g\|} - \beta \|g\| = 0$$

which implies that $\beta = \beta^* = \langle e_{n-1}, g \rangle / \|g\|^2$, and $\Delta e_{\max}(g) = \langle e_{n-1}, g \rangle^2 / \|g\|^2$. Maximization of $\Delta e(g)$ for a given g implies the minimization of $\|f - (f_{n-1} + \beta g)\|$ over β . Based on the set Γ , the expression $\|f - (f_{n-1} + \beta g)\|$ is minimized when $\Delta e_{\max}(g)$ is maximized over all $g \in \Gamma$. We define the span of Γ as the set of all linear combinations of the elements of Γ . It is

¹ $\|g\| = 0$ implies that $g(x) \equiv 0 \forall x \in X$.

well established that $\text{span}(\Gamma)$ is dense in the space of all square integrable functions or L^2 space [8], [38].

Theorem 1: Given that $\text{span}(\Gamma)$ is dense in L^2 and $\forall g \in \Gamma, 0 < \|g\| < b$ for some $b \in \mathfrak{R}$. If g_n is selected to maximize $\langle e_{n-1}, g \rangle^2 / \|g\|^2$, then $\lim_{n \rightarrow \infty} \|f - f_n\| = 0$.

Proof: It follows from Proposition 1 that

$$\|e_{n-1}\|^2 - \|e_n\|^2 = \frac{\langle e_{n-1}, g_n \rangle^2}{\|g_n\|^2} \quad (10)$$

which is bounded in the following discussion by zero. The boundary case, i.e., zero will occur when e_{n-1} , is orthogonal to g_n . $\text{Span}(\Gamma)$ is dense in L^2 ; thus, orthogonality will occur only when $e_{n-1} = 0$ [39, Lemma 3.3–7]. In that case, the sequence $\{\|e_n\|^2\}$ is already converged. In other words, $\{\|e_n\|^2\}$ is a strictly decreasing sequence and converges to zero. ■

The boundedness assumption of $\|g\|$ holds, because NCA uses a sigmoid transfer function for all hidden neurons. Theorem 1 leads to strong convergence, provided that a constructive algorithm in each incremental step can find g_n to maximize $\langle e_{n-1}, g \rangle^2 / \|g\|^2$. However, it is impractical to search g_n over the whole set Γ in each step. In practice, constructive algorithms search g_n over a smaller subspace, because as we have mentioned, the particular form of g_n is determined by how a hidden neuron is connected to the current ANN.

BP [32] minimizes the ANN error by adjusting its connection weights, i.e., by searching a g_n over a subspace, and NCA adds random noises to the connection weights for escaping from local minima; thus, it can be expected that NCA approximates such a g_n over the subspace that maximizes $\langle e_{n-1}, g \rangle^2 / \|g\|^2$. However, it is clear from (10) that the reduction of residual error will be close to zero if e_{n-1} is nearly orthogonal to g_n . Moreover, Theorem 1 implies that, in each step, e_n is orthogonal to the span $\{g_1, g_2, \dots, g_n\}$; otherwise, e_n will not be the minimum for the given g_1, g_2, \dots, g_n . This condition indicates that, if $g_{n+1} \approx g_n$, e_n will also be nearly orthogonal to g_{n+1} , resulting in $\langle e_n, g_{n+1} \rangle / \|g\|^2 \approx 0$. In other words, the residual error will not decrease when $g_{n+1} \approx g_n$. The proposed NCA detects this situation using a condition, as described by (5), of its layer-addition criterion that determines whether $g_{n+1} \approx g_n$. In this situation, one should extend the search space. The proposed NCA extends the search space by adding a new hidden layer with one neuron. The neuron of the new hidden layer gets inputs from all preceding layers (see Fig. 1); thus, it is not difficult to show that the new subspace contains the earlier subspace and produces a different g [40]. In fact, NCA expands the search space by creating a new layer whenever it detects $g_{n+1} \approx g_n$. Thus, it complies with Theorem 1 and, hence, converges.

F. Computational Complexity

We have already seen NCA adopt several techniques in its different stages. The computational complexities of these techniques and NCA, as a whole, are presented in this section.

1) *Creation of a New Training Set:* The proposed NCA creates a new training set by employing a similar concept used in AdaBoost [24], which maintains a probability distribution over a training set. Each time NCA trains a newly added hidden neuron, it changes the distribution by increasing the probabilities of misclassified examples of an existing ANN while decreasing

the probabilities of correctly classified examples. To update the probabilities, it requires $O(P_t)$ computations, where P_t is the number of training examples in a training set. Given a probability distribution, NCA selects P_t examples (with replacement) using at most $O(P_t \log_2 P_t)$ computations. It is because NCA first computes cumulative probabilities by a cost of $O(P_t)$ computations. Then, for each selection, NCA applies the binary search of a generated random number over the cumulative probabilities, which requires at most $O(\log_2 P_t)$ computations.

2) *Initial Training*: We use the standard BP algorithm [32] to train an ANN. Each epoch of BP takes $O(W)$ computations for one example, where W is the number of weights in the ANN. Thus, training P_t examples for τ epochs requires $O(\tau \times P_t \times W)$ computations.

3) *Stop ANN Construction*: The termination criterion in NCA for stopping ANN construction uses the training and the validation errors. The training error is computed as a part of training; thus, the stopping operation takes $O(P_v \times W)$ computations, where P_v denotes the number of examples in the validation set.

4) *Stop Initial Training*: This operation takes a constant $O(1)$ computation, because NCA stops the initial training based on the training error, which is a part of training and does not require any extra computations.

5) *Final Training*: All computations for the final training are the same as the initial training, except that the final training needs an extra cost of $O(W)$ computations for adding noise. Thus, the cost of the final training is $O(\tau \times P_t \times W) + O(W)$ i.e., $O(\tau \times P_t \times W)$.

6) *Stop Final Training*: This operation also takes a constant $O(1)$ computation as the stop initial training.

7) *Checking for Adding a Hidden Layer*: The proposed NCA uses (4) and (5) to add a hidden layer. It takes a constant computation $O(1)$ for evaluating (4) and $O(P_t)$ operations for evaluating (5), because the outputs of hidden neurons for each training example are stored during the training process. Thus, checking for adding a hidden layer takes $O(P_t)$ computations.

8) *Adding a Hidden Neuron*: It takes $O(i + \sum_{j=1}^{L-1} h_j)$ computations for the initialization of connection weights, where i is the number inputs, h_j is the number of neurons of the j th hidden layer, and L the current hidden layer. Note that $(i + \sum_{j=1}^{L-1} h_j) < W$.

9) *Adding a Hidden Layer*: The computation cost for adding a hidden layer is the same as adding a hidden neuron, except that we have L instead of $L - 1$. Thus, the computation cost of this step is $O(i + \sum_{j=1}^L h_j)$.

Based on the aforementioned discussion, it is clear that the leading computation requirement is training with a cost of $O(\tau \times P_t \times W)$ computations. Practically, $\tau \times W > \log_2 P_t$ and $P_t \geq P_v$; thus, the computation complexity of NCA is $O(\tau \times P_t \times W)$, which is the same order of training a fixed ANN architecture using BP [32].

IV. EXPERIMENTAL STUDIES

We evaluate the performance of NCA on several benchmark problems that were selected from the University of California–Irvine Machine Learning Repository, which are summarized in Table I. The selected problems have considerable diversities: some were easy (e.g., the cancer problem),

TABLE I
CHARACTERISTICS OF TEN BENCHMARK PROBLEMS. NOTE THAT THE FIRST EIGHT AND THE LAST TWO PROBLEMS ARE CLASSIFICATION AND APPROXIMATION PROBLEMS, RESPECTIVELY

Problem	Number of				
	input attributes	classes/ outputs	training examples	validation examples	testing examples
Cancer	9	2	350	175	174
Card	14	2	345	173	172
Heart	35	2	460	230	230
Gene	120	3	1588	794	793
Glass	9	6	107	54	53
Iris	4	3	75	38	37
Letter	16	26	1000	500	4000
Thyroid	21	3	3600	1800	1800
Building	14	/3	2104	1052	1052
Flare	24	/3	533	267	266

some were difficult (e.g., the glass problem), and some were large (e.g., the letter problem). Hence, the selected problems provide a good test bed for evaluating the performance of any algorithm on a set of problems with different characteristics. These problems were also widely used in many previous studies (e.g., [2], [4], [11], and [17]).

A. Experimental Setup

In the past, there were some criticisms toward ANN benchmarking methodologies and suggestions for improvement have been put forward [33], [41]. We used benchmark problems and followed the benchmarking methodologies to conduct experiments and presenting results. The same problems and methodologies were employed in many previous and recent studies (e.g., [2], [5], [7], [17], and [20]). The data sets of all problems were partitioned into the following three sets: 1) a training set; 2) a validation set; and 3) a testing set. The number of examples in these sets is shown in the Table I. For all problems except letter, the first P_1 examples were used for the training set, the following P_2 examples for the validation set, and the final P_3 examples for the testing set. To have a fair comparison with our previous work, for the letter problem, we randomly selected 1000, 500, and 4000 examples for the training, validation, and testing sets, respectively. In all experiments, we used one bias neuron with a fixed input +1 for the hidden and output layers. We employed a logistic sigmoid activation function, $f(x) = 1/(1 + \exp(-x))$, for hidden layers. For classification problems, the same activation function was also employed for the output layer. However, for approximation problems, we employed the identity function for the output layer.

The initial connection weights of ANNs were chosen between -0.5 and $+0.5$. The learning rate and momentum term of BP [32] were set to 0.10 and 0.8, respectively. As shown in Section III, NCA introduces three control parameters: 1) τ ; 2) ϵ_1 ; and 3) ϵ_2 . The value of τ was set to 20 for all problems except letter, which used a τ of 50. The thresholds ϵ_1 and ϵ_2 were set to $2e-03$ and 0.20, respectively. These values were chosen after a preliminary run.

TABLE II
PERFORMANCE OF NCA ON EIGHT CLASSIFICATION (THE FIRST EIGHT ROWS) AND TWO APPROXIMATION (THE LAST TWO ROWS) PROBLEMS. TER AND STE REPRESENT THE GENERALIZATION ABILITY OF PRODUCED ANNS FOR CLASSIFICATION AND APPROXIMATION PROBLEMS, RESPECTIVELY. THE RESULTS WERE AVERAGED OVER 50 INDEPENDENT RUNS. SD REPRESENTS STANDARD DEVIATION

Problem	Number of hidden layers		Number of hidden neurons		Number of epochs		TER /STE	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD
Cancer	1.02	0.14	2.08	0.27	293.3	30.80	0.91	0.24
Card	1.00	0.00	1.20	0.39	105.8	13.70	13.75	0.53
Gene	1.06	0.24	2.18	0.48	980.7	42.18	10.46	0.31
Glass	1.52	0.50	3.82	0.56	384.5	36.02	30.56	1.85
Heart	1.04	0.20	2.42	0.57	170.6	17.50	18.17	1.08
Iris	1.00	0.00	2.60	0.73	125.3	13.76	2.16	0.18
Letter	2.76	0.43	15.64	4.50	5530.8	277.34	19.20	1.47
Thyroid	1.36	0.48	4.86	1.16	401.7	30.14	1.50	0.21
Building	1.40	0.49	5.58	1.28	512.3	41.90	/0.45	/0.10
Flare	1.00	0.00	2.20	0.53	101.4	22.32	/0.51	/0.12

B. Experimental Results

Table II shows the results of NCA over 50 independent runs for different problems. For each run, moderate changes were made in the parameters value mentioned in the previous section. Testing error rate (TER) refers to the percentage of wrong classifications produced by trained ANNs on the testing set of a classification problem. The squared testing error (STE), which is computed according to (1), refers to the error produced by trained ANNs on the testing set of an approximation problem. The TER or STE represents the generalization ability of produced ANNs. Fig. 3 shows the training progress of ANNs for the glass and thyroid problems.

The effect of different techniques that were adopted in NCA is clearly visible from the results presented in Table II and Fig. 3. The following observations can be made based on these results.

- 1) It can be observed that NCA produces different-sized ANNs for different problems, and NCA spent a different number of epochs for producing ANNs (see Table II).
- 2) Our NCA produced complex ANN architectures for some problems and simple ANN architectures for other problems (see Table II). For the letter problem, for example, ANNs produced by NCA had, on the average, 2.76 hidden layers and 15.64 hidden neurons, whereas those produced for the iris problem had, on the average, 1.00 hidden layer and 2.60 hidden neurons. The letter problem was a 26-class problem, whereas iris was a three-class problem (see Table I). The complexity of these two problems can be understood from the TER and the number of training epochs (see Table II). The proposed NCA spent, on the average, 5530.8 epochs for the letter problem, which was larger compared to that spent for the iris problem. Furthermore, the TER for the letter problem was also larger (worse) compared to that for the iris problem.
- 3) The number of hidden layers in ANNs produced by NCA had, on the average, one for three problems and more than two for one problem. However, ANNs had, on the average, more than one and less than two hidden layers for six

problems. These results indicate that, even for the same problem, ANNs that were produced by NCA had a different number of hidden layers in different independent runs. In other words, the characteristics of a problem were not only the factor that determined the number of hidden layers in an ANN but also different parameters that were employed by a learning and an architecture determination algorithm. A similar argument could be drawn for the number of hidden neurons in ANNs. These results quite match our intuition that puts emphasis on determining the complete topological information of ANNs for solving different problems.

- 4) It is shown in Fig. 3 that the training error gradually decreased as a training process progressed and hidden neurons added. When the error reduction slowed down, the addition of hidden neurons accelerated such a reduction. In fact, there is no sign of trapping at serious local optima during the entire training process of ANNs. Although the theoretical analysis in Section III-E confirms the convergence of NCA, Fig. 3 gives its empirical evidence.

Recently, a receiver-operating characteristics (ROC) graph has been used, aside from classification accuracy to a learning mechanism, to examine its performance in the machine learning community. The main advantage of the ROC graph is that it is independent of class distribution; thus, it is very useful for problems with unbalanced classes. The ROC graph is a plot where the false positive rate is on the x -axis and the true positive rate is on the y -axis. The point (0,1) is the perfect classifier: it correctly classifies all positive cases and negative cases. In general, the upper left region indicates the good performance of ANNs. A detailed description of the ROC graph can be found in [42]. Fig. 4 shows the ROC graphs for the cancer and heart problems. This figure also illustrates the good performance of NCA.

1) *Effect of Different Training Sets:* A set of new experiments was carried out to observe the effect of different training sets used in NCA. The setup of these experiments was exactly the same as those previously described. The only difference was that NCA trained here all hidden neurons using the same training set. We call this variant of NCA as NCAST, which is equivalent to NCA without functional adaptation.

Table III presents the results of NCAST over 50 independent runs. The comparison of results in Tables II and III indicated that NCAST used more epochs and larger ANN architectures than NCA. This step is reasonable, because NCA trained all hidden neurons of an ANN using different training sets, whereas NCAST trained all hidden neurons using the same training set. Hidden neurons could not communicate each other during training; thus, they might learn the same information from the same training set. It is natural to require more computational resources, i.e., large architectures and many training epochs, when hidden neurons learn redundant information for solving a given problem.

To gain a better understanding about the performance difference between NCA and NCAST, we measured the functional dissimilarity between different pairs of hidden neurons in an ANN. We measured the functional dissimilarity between two hidden neurons based on the original training set using (5). There are C_2^M numbers of functional dissimilarities for an ANN with M hidden neurons. In our experiments, both NCA

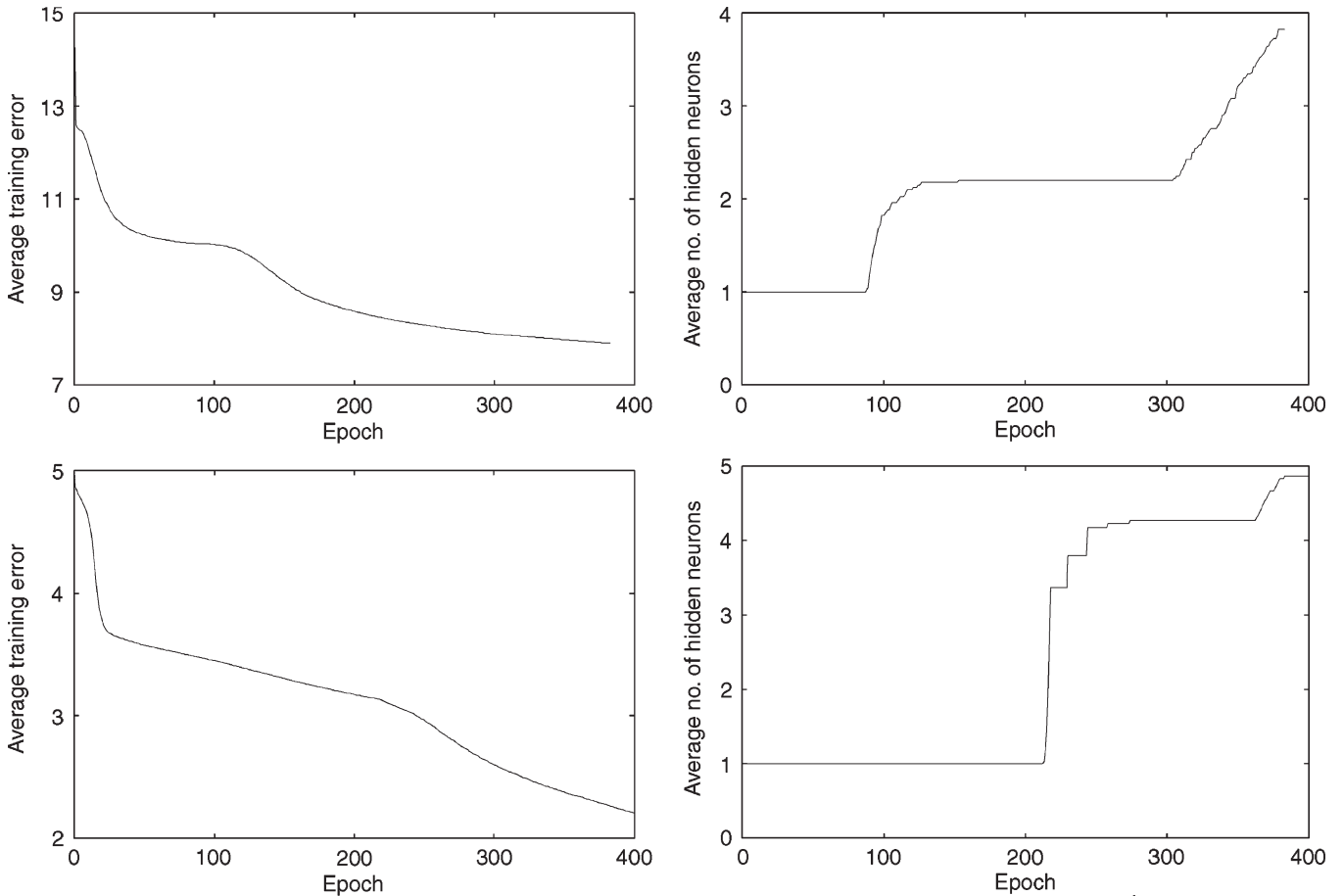


Fig. 3. Training progress of NCA for the (top row) glass and (bottom row) thyroid problems. All results have been averaged over 50 runs.

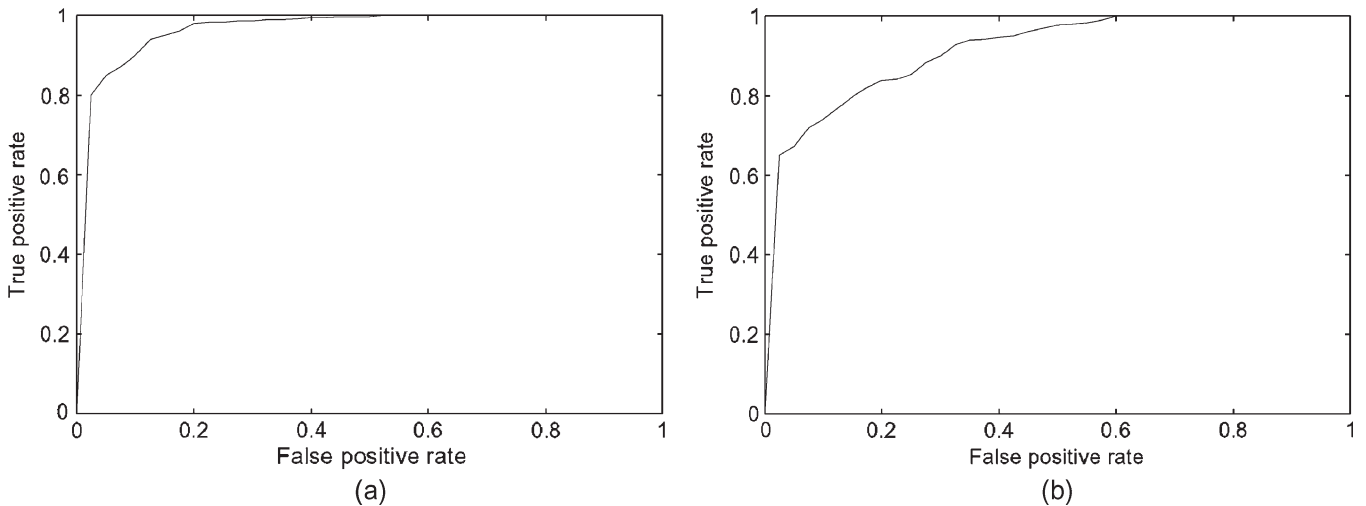


Fig. 4. ROC graphs for the (a) cancer and (b) heart problems. All results have been averaged over 50 runs.

and NCAST were allowed to add six hidden neurons within 450 training epochs. These values were chosen based on our experience in our previous experiments. The reason for choosing the same value is to make a fair comparison.

Fig. 5 shows the functional dissimilarity of different pairs of hidden neurons in an ANN trained by NCA and NCAST for the thyroid problem. A similar phenomenon was observed for other problems. This figure also shows the number of hidden layers against the hidden neuron number. It is shown in Fig. 5

that some degree of regularity is apparent in the functionality of hidden neurons trained by NCA. This regularity is due to training different hidden neurons by different training sets that were created by the Adaboost algorithm [24]. The dissimilarity between the training sets of hidden neurons h_m and h_{m+1} , for $1 \leq m \leq M$, is less compared to, for example, those of h_m and h_{m+1+i} , for $i \geq 1$ and $(m + 1 + i) \leq M$. This instance is why the dissimilarity between the functionality of h_m and h_{m+1} is small compared to that of h_m and h_{m+1+i} . Although

TABLE III

PERFORMANCE OF NCAST ON FOUR CLASSIFICATION (THE FIRST FOUR ROWS) AND ONE APPROXIMATION (THE LAST ROW) PROBLEMS. TER AND STE REPRESENT THE GENERALIZATION ABILITY OF PRODUCED ANNS FOR CLASSIFICATION AND APPROXIMATION PROBLEMS, RESPECTIVELY. ALL RESULTS WERE AVERAGED OVER 50 INDEPENDENT RUNS. SD REPRESENTS STANDARD DEVIATION

Problem	Number of hidden layers		Number of hidden neurons		Number of epochs		TER /STE	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD
Card	1.00	0.00	1.28	0.54	110.3	21.26	13.75	0.60
Gene	1.12	0.33	3.80	0.93	1050.9	37.18	12.48	0.24
Glass	1.60	0.47	4.04	0.49	410.5	42.80	31.69	2.10
Letter	3.02	0.52	17.74	6.50	6021.3	312.74	21.96	1.94
Thyroid	1.56	0.67	6.12	0.72	460.8	42.52	3.19	0.17
Building	1.52	0.58	6.94	1.47	590.7	38.00	10.62	10.12

there are many maxima and minima in the functionality of hidden neurons trained by NCAST, no definite regularity is visible; rather, the histograms consist of some ups and downs that indicate the random learning strategy of hidden neurons. It is also shown that hidden neurons trained by NCAST were functionally less dissimilar compared to that trained by NCA (see Fig. 5). Accordingly, ANNs that were produced by NCAST had more hidden layers (see Tables II and III). This result is because NCAST (or NCA) added hidden layers when the functionality of hidden neurons became similar.

The positive effect of compact architectures and small training epochs can be observed on the TER/STE of produced ANNs. The TER/STE achieved by NCA for different problems was lower compared to those achieved by NCAST (see Tables II and III). The t -test based on the number of hidden neurons, epochs, and TER/STE indicated that NCA was significantly better than NCAST at a 95% confidence level, with the exception of the card problem. The performance of NCA and NCAST was found similar for the card problem. As shown in Tables II and III, an ANN with one or slightly more hidden neurons could solve the card problem. Both NCA and NCAST started training on the original training data for an ANN with one hidden neuron; thus, the strategy employed by NCA to create a new training set could barely be utilized. This instance is the main reason for the similar performance of NCA and NCAST for the card problem. In short, the performance difference between NCA and NCAST indicates that the incorporation functional adaptation with architectural adaptation, which we employed in NCA, can greatly improve the performance of ANNs.

2) *Effect of the Layer-Addition Criterion*: The purpose of this section is to evaluate the effect of the hidden-layer-addition criterion used in NCA. To understand the effect, we performed a set of new experiments using NCA without the layer-addition criterion. This variant of NCA is dubbed NCAWL. It can be observed in Table II that ANNs produced by NCA for the glass, letter, thyroid, and building problems had more than one hidden layers, whereas those produced for the cancer, card, gene, heart, iris, and flare problems had one or slightly more hidden layers. It is worth mentioning that the only difference between NCA and NCAWL is that the former approach could produce ANNs with one or more hidden layers, whereas the latter one could produce ANNs with only one hidden layer. Thus, we applied

NCAWL to the glass, letter, thyroid, and building problems so that the effect of the layer-addition criterion could easily be understood.

Table IV presents the results of NCAWL on four problems over 50 independent runs. The positive effect of the layer-addition criterion can be observed when we compare the results presented in Table IV with those presented in Table II. For example, NCAWL spent more epochs, produced larger ANN architectures, and achieved larger TERs/STEs compared to those of NCA. The t -test based on the number of hidden neurons, number of epochs, and TER/STE indicated that NCA was significantly better than its counterpart NCAWL at a 95% confidence level.

To gain a better understanding about the layer-addition criterion, we investigated the necessity of adding new hidden layers. Fig. 6 shows the functional dissimilarity between two consecutive hidden neurons h_m and h_{m+1} , for $1 \leq m \leq M$, of an ANN trained by NCA and NCAWL for the letter problem. Unlike Fig. 5, we computed the functional dissimilarity on the training set of h_{m+1} , which was used in (5) for adding a new hidden layer. It is shown in Fig. 6 that the functional dissimilarity gradually decreased as NCAWL added all hidden neurons in the same hidden layer. This dissimilarity was the same for both NCAWL and NCA until the later approach added the ninth hidden neuron in the new, i.e., the second, hidden layer. When NCA added the ninth hidden neuron, the functional dissimilarity increased but again started decreasing. The reason for increasing the functional dissimilarity is due to processing different information by the neuron. Consider two hidden neurons h_m and h_{m+1} that reside in two different hidden layers l and $l+1$, respectively. For a particular training example, the h_m neuron receive information from all preceding layers, i.e., up to layer $(l-1)$, whereas the h_{m+1} neuron receive information up to layer l . Two neurons process different information; thus, they will exhibit different functionality for the same training example. The results in Fig. 6 indicate the necessity of adding hidden layers to increase the functional dissimilarity among hidden neurons in the ANN. The hidden-layer-addition capability made NCA better compared to NCAWL, which tried to solve all problems using ANNs with only one hidden layer.

Now let us look at what happens when we compare the results in Table IV with those in Table III. In terms of TER/STE, it is clearly visible that neither NCAWL nor NCAST is found better for all the four problems that we compare here. The NCAST algorithm is found to be significantly better than NCAWL for the glass problem, whereas NCAWL is found to be significantly better than NCAST for the thyroid problem. For the letter and building problems, the TER/STE of both NCAST and NCAWL was found similar. These comparisons again indicate that neither architectural adaptation nor functional adaptation alone is sufficient for efficiently solving problems using ANNs.

3) *Effect of Control Parameters*: We have seen in Section III that NCA introduces three control parameters (i.e., τ , ϵ_1 , and ϵ_2) in its adaptation process. This case is usual in the field (e.g., [2], [5], [7], and [17]). As mentioned in Section III, τ defines the number of training epochs for the initial partial training of a hidden neuron. The parameter ϵ_1 measures the improvement of training error, whereas ϵ_2 measures the functional dissimilarity between any two hidden neurons. We can choose any positive integer value for τ and any real value for ϵ_1 and ϵ_2 .

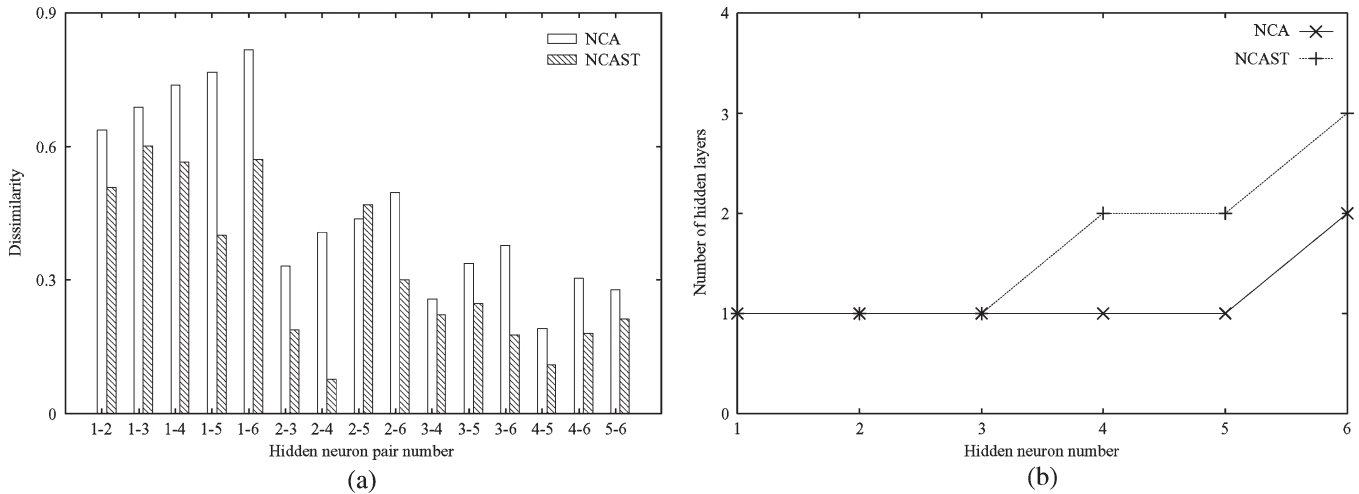


Fig. 5. (a) Functional dissimilarities among the hidden neurons' pairs in an ANN trained by NCA and NCAST. (b) The addition of hidden neurons in the hidden layer(s) by NCA and NCAST. The presented result is the single run of NCA and NCAST for the thyroid problem.

TABLE IV
PERFORMANCE OF NCAWL ON THREE CLASSIFICATION (THE FIRST THREE ROWS) AND ONE APPROXIMATION (THE LAST ROW) PROBLEMS. TER AND STE REPRESENT THE GENERALIZATION ABILITY OF PRODUCED ANNs FOR CLASSIFICATION AND APPROXIMATION PROBLEMS, RESPECTIVELY. ALL RESULTS WERE AVERAGED OVER 50 INDEPENDENT RUNS. SD REPRESENTS STANDARD DEVIATION

Problem	Number of hidden neurons		Number of epochs		TER/STE	
	Mean	SD	Mean	SD	Mean	SD
Glass	5.10	1.20	420.9	36.50	33.20	1.98
Letter	18.40	4.85	5790.3	401.32	21.68	2.32
Thyroid	5.96	0.88	490.2	72.54	2.21	0.20
Building	6.20	1.50	543.5	42.30	/0.77	/0.11

Although a very small value for τ may undertrain a hidden neuron, a very large value may overtrain the neuron. In any case, NCA deals with this problem by introducing final training and keeping the weights of a hidden neuron fixed after final training. If the value of ϵ_1 is set very large but that of ϵ_2 is very small, ANNs that were produced by NCA will have a large number of hidden layers and a small number of hidden neurons [see (4) and (5)]. An opposite scenario will happen, i.e., the number of hidden layers and of hidden neurons would be small and large, respectively, for a very small ϵ_1 and a very large ϵ_2 . That is, the value for ϵ_1 and ϵ_2 chosen in an opposite direction will affect either the number of hidden layers or the number of hidden neurons but not both. This instance is a desirable feature in the sense that ANNs that were produced by NCA will have a similar number of connections for different ϵ_1 s and ϵ_2 s.

If there are φ possible choices, we can select a set of values for three control parameters from a combination of φ^3 values. Thus, it is not easy to select values for τ , ϵ_1 , and ϵ_2 without looking at their relationship. We have already established the relationship between ϵ_1 and ϵ_2 . Now, we shall look at the relationship between τ and ϵ_1 (or ϵ_2). For example, if the value of τ is chosen small, the training error will be reduced by a small amount due to insufficient training, and the functional dissimilarity between hidden neurons will also be small for the same reason. This result means that, if the value of τ is chosen small, moderate, or large, the value of ϵ_1 (or ϵ_2) will also be

chosen in the same manner. Note that the value of ϵ_1 will be chosen in the opposite direction of ϵ_2 . In general, we can say that τ , ϵ_1 , and ϵ_2 will have less effect on NCA, provided that they are chosen in accordance with the aforementioned guidelines. The following experimental results support our intuition.

We have chosen the glass, heart, and letter problems for empirical studies here. As shown in Tables II–IV, these three problems were the most challenging among all other problems that we tested in this paper. The use of three problems from a set of ten problems helps us limit the size of this paper. In experiments, τ was chosen in the range of 10 to 50 for the heart and glass problems and in the range of 30 to 80 for the letter problem. The thresholds ϵ_1 and ϵ_2 were chosen in the range of $1e-04$ to $3e-03$ and 0.05 to 0.30, respectively. The value of other parameters in NCA was chosen to be the same as mentioned in Section IV-A. Tables V–VII show the performance of NCA with different parameter values for the glass, heart, and letter problems. It is shown that, when the values of τ , ϵ_1 , and ϵ_2 were chosen to be small, medium, or large, the generalization ability (i.e., TER) of ANNs produced by NCA was not much affected. However, when the value of τ was chosen to be large but the values of ϵ_1 and ϵ_2 were chosen to be small, the generalization ability was much affected.

C. Comparisons With Other Work

This section presents the comparison between NCA and other algorithms based on the experimental results of different problems. Although there are many architecture determination algorithms in the literature that could be compared, it is not feasible to conduct an exhaustive comparison with all such algorithms. The aim of our experimental comparison here is to evaluate and understand the strengths and weaknesses of NCA for different problems. NCA uses a constructive approach in determining network architectures; thus, few similar and recent algorithms are chosen for comparison. These algorithms are listed as follows:

- 1) constructive feedforward neural network (CFNN) [17];
- 2) optimization methodology for neural network (OMNN) [20];
- 3) cascade correlation learning architecture (CCLA) [14];

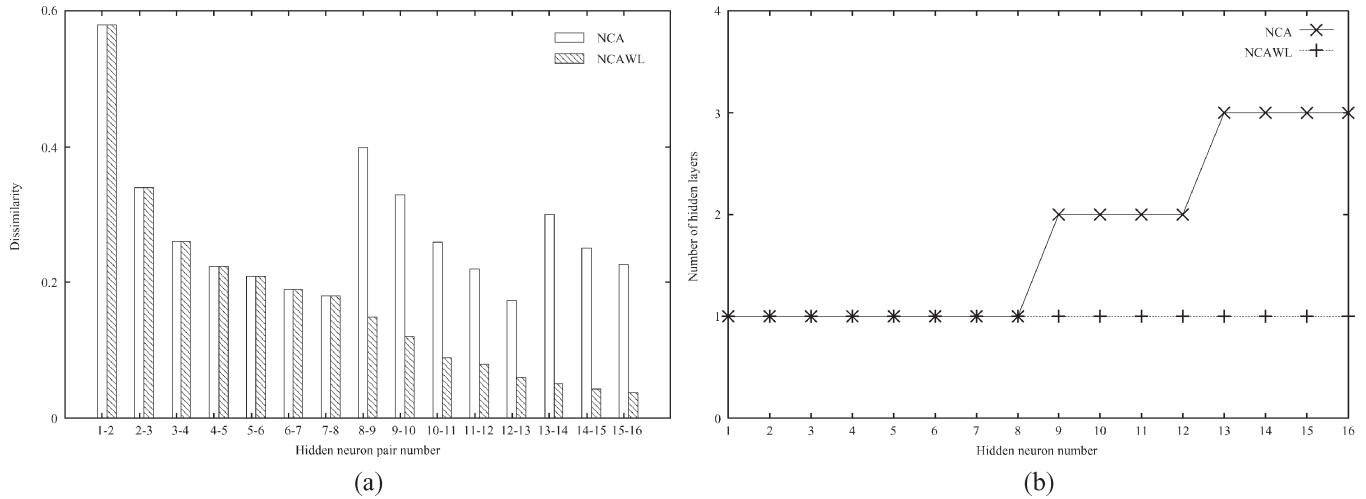


Fig. 6. (a) Functional dissimilarities among the hidden neurons’ pairs in an ANN trained by NCA and NCAWL. (b) The addition of hidden neurons in the hidden layer(s) by NCA and NCAWL. The presented result is the single run of NCA and NCAWL for the letter problem.

TABLE V
PERFORMANCE OF NCA WITH DIFFERENT PARAMETER VALUES FOR THE GLASS PROBLEM. TER REPRESENTS THE GENERALIZATION ABILITY OF PRODUCED ANNs FOR THE GIVEN CLASSIFICATION PROBLEM. ALL RESULTS WERE AVERAGED OVER 50 INDEPENDENT RUNS. SD REPRESENTS STANDARD DEVIATION

Parameter value	Number of hidden layers		Number of hidden neurons		Number of epochs		TER	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD
τ ϵ_1 ϵ_2								
10 1e-03 0.10	1.48	0.65	4.04	0.55	370.5	30.52	31.04	1.52
30 5e-03 0.25	1.58	0.50	3.78	0.54	398.2	44.74	30.20	0.95
40 9e-03 0.30	1.56	0.64	3.70	0.58	420.3	24.30	31.86	1.22
50 1e-04 0.05	1.52	0.58	3.58	0.83	480.8	30.02	33.02	1.74

TABLE VI
PERFORMANCE OF NCA WITH DIFFERENT PARAMETER VALUES FOR THE HEART PROBLEM. TER REPRESENTS THE GENERALIZATION ABILITY OF PRODUCED ANNs FOR THE GIVEN CLASSIFICATION PROBLEM. ALL RESULTS WERE AVERAGED OVER 50 INDEPENDENT RUNS. SD REPRESENTS STANDARD DEVIATION

Parameter value	Number of hidden layers		Number of hidden neurons		Number of epochs		TER	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD
τ ϵ_1 ϵ_2								
10 1e-03 0.10	1.10	0.30	2.62	0.78	160.8	16.58	17.98	1.20
30 5e-03 0.25	1.02	0.14	2.38	0.63	181.9	18.20	17.64	0.98
40 9e-03 0.30	1.00	0.00	2.54	0.54	182.5	19.34	17.68	1.04
50 1e-04 0.05	1.02	0.14	2.34	0.60	255.7	30.62	19.54	1.32

- 4) constructive neural network design algorithm (CNDA) [13];
- 5) modified CCLA (MCCLA) [28];
- 6) feedforward neural network constructive algorithm (FNNCA) [43].

All these algorithms, except CNDA and OMNN, employed a constructive approach in determining ANN architectures. CNDA employed a hybrid constructive and pruning approaches, whereas OMNN employed a hybrid simulated annealing [44] and tabu search [45] methods.

The BP algorithm [32] was used in NCA, CNDA, MCCLA, and OMNN for training ANNs, whereas the quasi-Newton method [43] was used in FNNCA. CCLA and CFNN used the quickprop method [46] for training ANNs. It has been known

TABLE VII
PERFORMANCE OF NCA WITH DIFFERENT PARAMETER VALUES FOR THE LETTER PROBLEM. TER REPRESENTS THE GENERALIZATION ABILITY OF PRODUCED ANNs FOR THE GIVEN CLASSIFICATION PROBLEM. ALL RESULTS WERE AVERAGED OVER 50 INDEPENDENT RUNS. SD REPRESENTS STANDARD DEVIATION

Parameter value	Number of hidden layers		Number of hidden neurons		Number of epochs		TER	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD
τ ϵ_1 ϵ_2								
30 1e-03 0.10	2.84	0.54	17.22	5.98	5480.5	225.70	18.78	1.64
40 5e-03 0.25	2.70	0.46	16.52	5.61	5563.2	260.38	19.42	1.82
60 9e-03 0.30	2.72	0.50	15.84	4.18	5612.8	301.86	19.88	1.58
80 1e-04 0.05	2.50	0.48	15.04	3.82	5822.4	284.54	21.94	1.60

TABLE VIII
COMPARISON BETWEEN NCA, CNDA [13], CCLA [14], CFNN [17], MCCLA [28], AND FNNCA [43] ON THE CANCER PROBLEM. TER REPRESENTS THE GENERALIZATION ABILITY OF PRODUCED ANNs FOR THE GIVEN CLASSIFICATION PROBLEM. THE RESULTS OF NCA AND CCLA WERE AVERAGED OVER 50 INDEPENDENT RUNS, WHEREAS THEY WERE AVERAGED OVER 30 INDEPENDENT RUNS FOR CNDA AND OMNN. THE RESULTS OF MCCLA AND FNNCA WERE THE BEST OF 5 AND 50 INDEPENDENT RUNS, RESPECTIVELY. SD REPRESENTS STANDARD DEVIATION AND ‘-’ MEANS NOT AVAILABLE

Algorithm	Number of hidden neurons		Number of epochs		TER	
	Mean	SD	Mean	SD	Mean	SD
	NCA	2.08	0.27	293.3	30.80	0.91
CCLA	5.18	2.05	-	-	1.95	0.38
CFNN	2.00	-	-	-	3.30	-
CNDA	3.50	-	451.6	-	1.10	-
MCCLA	4.00	-	401.0	-	1.10	-
FNNCA	3.00	-	-	-	1.40	-

that quasi-Newton and quickprop are faster than BP. To make a fair comparison, we compared NCA with other algorithms in terms of the number of hidden neurons and TER/STE. However, the number of training epochs was also compared when other algorithms used BP for training ANNs.

Tables VIII–XI present the results of NCA and other algorithms for several classification and approximation problems. It is shown that NCA achieved the smallest TER/STE for eight

TABLE IX

COMPARISON BETWEEN NCA AND CCLA [14] ON FOUR CLASSIFICATION (THE FIRST FOUR ROWS) AND TWO APPROXIMATION (THE LAST TWO ROWS) PROBLEMS. TER AND STE REPRESENT THE GENERALIZATION ABILITY OF PRODUCED ANNs FOR CLASSIFICATION AND APPROXIMATION PROBLEMS, RESPECTIVELY. THE RESULTS OF BOTH NCA AND CCLA WERE AVERAGED OVER 50 INDEPENDENT RUNS. SD REPRESENTS STANDARD DEVIATION

Problem	Algorithm	Number of hidden neurons		TER/STE	
		Mean	SD	Mean	SD
		Card	NCA	1.20	0.49
	CCLA	1.07	0.25	13.72	0.43
Gene	NCA	2.18	0.39	10.46	0.31
	CCLA	2.73	1.19	13.38	0.47
Glass	NCA	3.82	0.56	30.56	1.85
	CCLA	8.07	5.19	34.76	5.88
Heart	NCA	2.42	0.58	18.17	1.08
	CCLA	2.64	1.17	19.89	1.58
Building	NCA	5.58	1.28	/0.45	/0.12
	CCLA	9.27	3.73	/0.82	/0.23
Flare	NCA	2.20	0.53	/0.51	/0.10
	CCLA	2.63	0.67	/0.53	/0.10

TABLE X

COMPARISON BETWEEN NCA, CNNDa [13], AND MCCLA [28] ON THE LETTER PROBLEM. TER REPRESENTS THE GENERALIZATION ABILITY OF PRODUCED ANNs FOR THE GIVEN CLASSIFICATION PROBLEM. THE RESULTS OF NCA WERE AVERAGED OVER 50 INDEPENDENT RUNS, WHEREAS THEY WERE AVERAGED OVER INDEPENDENT 30 RUNS FOR CNNDa. THE RESULTS OF MCCLA WERE THE BEST RESULTS OF FIVE INDEPENDENT RUNS

Algorithm	Number of hidden neurons	Number of epochs	TER
NCA	15.64	5530.8	19.20
CNNDa	19.80	5913.5	23.00
MCCLA	36.00	7013.0	26.87

TABLE XI

COMPARISON BETWEEN NCA, CCLA [14], AND OMNN [20] ON THE THYROID PROBLEM. TER REPRESENTS THE GENERALIZATION ABILITY OF PRODUCED ANNs FOR THE GIVEN CLASSIFICATION PROBLEM. THE RESULTS OF NCA AND CCLA WERE AVERAGED OVER 50 INDEPENDENT RUNS, WHEREAS THEY WERE AVERAGED OVER 30 INDEPENDENT RUNS FOR OMNN. SD REPRESENTS STANDARD DEVIATION AND ‘-’ MEANS NOT AVAILABLE

Algorithm	Number of hidden neurons		TER	
	Mean	SD	Mean	SD
	NCA	4.86	1.15	1.50
CCLA	25.04	8.71	3.03	1.15
OMNN	6.39	-	7.30	-

TABLE XII

COMPARISON BETWEEN NCA, VNP [2], OBS [3], OMNN [20], AND OBD [47] ON THE IRIS PROBLEM. TER REPRESENTS THE GENERALIZATION ABILITY OF PRODUCED ANNs FOR THE GIVEN CLASSIFICATION PROBLEM. THE RESULTS OF NCA AND OMNN WERE AVERAGED OVER 50 AND 30 INDEPENDENT RUNS, RESPECTIVELY, WHEREAS THEY WERE NOT MENTIONED IN [2] FOR VNP, OBS, AND OBD

Algorithm	Number of hidden neurons	TER
NCA	2.60	2.16
VNP	2.00	2.30
OBS	4.00	2.0
OMNN	2.65	4.60
OBD	4.00	2.0

two key components: 1) training hidden neurons with different training sets and 2) adding new hidden layers based on hidden neurons’ functionality. This case is the main reason for the better performance of CCLA over NCA for the card problem. In addition, CCLA used direct connections from the input layer to the output layer. This result helped CCLA in solving the card problem with a little bit smaller number of hidden neurons.

Direct comparison with other algorithms using statistical tests is impossible, unless the results of each independent run are available. Such results for CCLA are available through an anonymous file transfer protocol from ftp.ira.uka.de in directory/pub/neuron as file nndata.tar.gz, and for CNNDa and MCCLA, they are available from our previous study [13]. We therefore conducted *t*-test to assess the significance in performance difference between NCA and CCLA/CNNDa/MCCLA. The *t*-test, based on the number of hidden neurons and TER/STE, indicates that NCA was significantly better than CCLA at a 95% confidence level, except for the card and flare problems. Both NCA and CCLA showed similar performance in terms of the number of hidden neurons and TER of the card problem. These two algorithms also showed similar performance when compared based on the STE of the flare problem. However, NCA was found to be better than CCLA based on the number of hidden neurons of the flare problem. The *t*-test, based on the number of hidden neurons, number of epochs, and TER, indicated that NCA was significantly better than CNNDa and MCCLA at a 95% confidence level for the two problems that we compared here.

We have not yet compared NCA with any algorithm based on pruning and evolutionary approaches. It is interesting to compare the performance of NCA with those algorithms. We therefore compared the performance of NCA with that of variance nullity pruning (VNP) [2], optimal brain surgeon (OBS) [3], EPNet [7] and optimal brain damage (OBD) [47]. The algorithms VNP, OBS, and OBD use a pruning approach in determining the number of hidden neurons of single-hidden-layered ANNs, whereas EPNet uses an evolutionary approach. Tables XII and XIII present the results of the aforementioned algorithms along with NCA. The results of NCA and EPNet were averaged over 50 and 30 independent runs, respectively, whereas they were not mentioned in [2] for VNP, OBS, and OBD. It is shown that NCA performed better than EPNet both in terms of hidden neurons and TER. However, NCA performed better than OBS and OBD in terms of hidden neurons, whereas OBS and OBD performed better than NCA in terms of TER.

out of nine problems. The proposed NCA also produced more compact network architectures compared to other algorithms for most of the problems. The performance of CCLA was found better than NCA for the card problem. Both NCA and CCLA used ANNs with, on the average, about one hidden neuron for the card problem; thus, NCA could barely utilize its

TABLE XIII
COMPARISON BETWEEN NCA AND EPNet [7] ON THE CANCER AND THYROID PROBLEMS. TER REPRESENTS THE GENERALIZATION ABILITY OF PRODUCED ANNS FOR THE GIVEN CLASSIFICATION PROBLEMS. THE RESULTS OF NCA WERE AVERAGED OVER 50 INDEPENDENT RUNS, WHEREAS THEY WERE AVERAGED OVER 30 INDEPENDENT RUNS FOR EPNet. SD REPRESENTS STANDARD DEVIATION

Problem	Algorithm	Number of hidden neurons		TER	
		Mean	SD	Mean	SD
		Cancer	NCA	2.08	0.27
	EPNet	2.00	1.10	1.37	0.93
Thyroid	NCA	4.86	1.15	1.50	0.21
	EPNet	5.90	2.40	2.12	0.22

The VNP algorithm performed better than NCA in terms of hidden neurons, whereas NCA performed better than VNP in terms of TER.

D. Discussion

This section briefly explains why the performance of NCA is better than other algorithms that we compared in Tables VIII–XI. There are a number of possible reasons for NCA's better performance.

First, NCA emphasizes on determining the complete topological information of an ANN architecture for solving problems. This algorithm therefore determines not only the number of hidden layers in ANNs but also the number of neurons in each hidden layer. In contrast, VNP [2], OBS [3], EPNet [7], CFNN [17], OMNN [20], CNNDa [13], FNNCA [43], and OBD [47] determine the number of hidden neurons for fixed hidden-layered ANNs. The other two algorithms, i.e., CCLA [14] and MCCLA [28], determine the number of hidden layers in ANNs with a fixed number of neurons in each hidden layer. It is clear, based on the characteristics of different problems and their TERs or STEs, that the complexity of all problems are not the same (see Table I and Tables VIII–XI). Our experimental results revealed that, even for the same problem, the number of hidden layers and of hidden neurons could be different due to the effect of different parameters involved in the architecture determination process (see Table II). They also revealed that the use of a fixed number of hidden layers was not beneficial for solving different problems (see Table III).

Second, NCA uses a training strategy that trains each hidden neuron in ANNs with a different training set. This training strategy encourages and forces each hidden neuron to work on unsolved parts of the training data. Although CFNN [17], CCLA [14], CNNDa [13], FNNCA [43], and CFNN [17] add hidden neurons one by one, with the aim of solving the unsolved parts of the training data, they train all hidden neurons using the same training data. When hidden neurons are added one by one, the neurons, which were added and trained earlier, could solve some parts of the training data. It is therefore natural to train a new hidden neuron on the remaining unsolved parts of the training data. The benefit of this approach is that each hidden neuron can concentrate on its own tasks; therefore, it can more efficiently handle the complex portion of the task. Our empirical studies reveal that such a training scheme is

beneficial for producing compact network architectures with a good generalization ability (see Tables II and III).

CNNDa [13] uses two phases: 1) an additive phase for determining the number of hidden neurons in ANNs roughly and 2) a pruning phase for refining the determination, i.e., for the removal of possible irrelevant connections and/or neurons. The aim of using the pruning phase is to produce compact network architectures. The problem with this approach is that the execution of the pruning phase requires some training epochs. This instance may overfit the training data due to overtraining, or the training algorithm may produce large connection weights. These two factors are not suitable for the generalization ability of ANNs [48]. CFNN used a different activation function, with the aim of producing ANNs with functionally different neurons. However, it is not known whether each added neuron is focused on the unsolved parts of the training data. The other two algorithms, i.e., CCLA [14] and FNNCA, do not use any scheme to produce compact ANN architectures.

The third reason is the effect of the layer-addition criterion. CCLA [14], MCCLA, [28] and CNNDa use a criterion based on the training error to add new hidden layers in ANNs. As aforementioned, the training error reduces when a hidden neuron is added in a new hidden layer or in an existing hidden layer. In other words, the error reduction does not give the precise information for the necessity of adding hidden layers. The proposed NCA therefore uses hidden neurons' functionality in association with the training error in the hidden-layer-addition criterion.

V. CONCLUSION

The automatic determination of ANNs' architecture is one of the most important and discussed issues in the neural network community. This paper has described a new constructive approach, i.e., NCA, to automatically determine ANNs' architecture. The idea behind NCA is to put more emphasis on complete architectural adaptation and functional adaptation rather than only partial architectural adaptation. A constructive approach is better suited due to its simplicity for handling such a multiobjective adaptation scheme. It also helps in avoiding the initialization problem suffered by other approaches.

A number of techniques have been adopted in NCA to achieve architectural adaptation and functional adaptation. For example, the proposed approach determines not only the number of hidden neurons in an ANN but also the number of neurons in each hidden layer to achieve complete architectural adaptation. This approach freezes the connection weights of a previously added hidden neuron and creates a new training set when a new neuron is added to the ANN. The new training set is created in such a way that it emphasizes on unsolved parts of the training data. The hidden-layer-addition criterion of NCA incorporates the functionality of hidden neurons' along with training error. NCA employs the aforementioned techniques to achieve functional adaptation during architectural adaptation.

The extensive experiments in this paper have been carried out to evaluate how well NCA performed on different problems compared with other algorithms. In almost all cases, NCA outperformed the others. However, our experimental study appeared to have revealed a weakness of NCA in dealing with the card problem, which can be solved by an ANN with about one hidden neuron. It is found that training hidden neurons with

different training sets do not work well for solving problems with a small-sized ANN. It would be interesting to more rigorously analyze NCA in the future to gain more insights into when NCA is most likely to perform well and for what kind of problems. In its current implementation, NCA has few user-specified parameters, although this case is not unusual in the field. These parameters, however, are not very sensitive to moderate changes. One of the future improvements to NCA would be to reduce the number of parameters or to make them adaptive. In addition, the use of a different activation functions in NCA for each hidden neuron in ANNs would also be an interesting future research topic.

ACKNOWLEDGMENT

The authors would like to thank the anonymous associate editor and reviewers for their constructive comments.

REFERENCES

- [1] T. Y. Kwok and D. Y. Yeung, "Constructive algorithms for structure learning in feedforward neural networks for regression problems," *IEEE Trans. Neural Netw.*, vol. 8, no. 3, pp. 630–645, May 1997.
- [2] A. P. Engelbrecht, "A new pruning heuristic based on variance analysis of sensitivity information," *IEEE Trans. Neural Netw.*, vol. 12, no. 6, pp. 1386–1399, Nov. 2001.
- [3] B. Hassibi and D. G. Stork, "Second-order derivatives for network pruning: Optimal brain surgeon," in *Advances in Neural Information Processing Systems*, vol. 5, C. Lee, S. Hanson, and J. Cowan, Eds. San Mateo, CA: Morgan Kaufmann, 1993, pp. 164–171.
- [4] Md. M. Islam, Md. A. Sattar, Md. F. Amin, X. Yao, and K. Murase, "A new adaptive merging and growing algorithm for designing artificial neural networks," *IEEE Trans. Syst., Man, Cybern. B: Cybern.*, vol. 39, no. 3, pp. 705–722, Jun. 2009.
- [5] I. Rivals and L. Personnaz, "Neural-network construction and selection in nonlinear modeling," *IEEE Trans. Neural Netw.*, vol. 14, no. 4, pp. 804–819, Jul. 2003.
- [6] S.-J. Han and S.-B. Cho, "Evolutionary neural networks for anomaly detection based on the behavior of a program," *IEEE Trans. Syst., Man, Cybern. B: Cybern.*, vol. 36, no. 3, pp. 559–570, Jun. 2006.
- [7] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Trans. Neural Netw.*, vol. 8, no. 3, pp. 694–713, May 1997.
- [8] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, 1989.
- [9] Md. A. Sattar, Md. M. Islam, and K. Murase, "A new constructive algorithm for designing and training artificial neural networks," in *Proc. 14th Int. Conf. Neural Inf. Process.*, Kitakyushu, Japan, Nov. 13–16, 2007, pp. 317–327.
- [10] H. Larochelle, D. Erhan, A. C. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *Proc. 24th Int. Conf. Mach. Learn.*, Corvallis, OR, Jun. 20–24, 2007, pp. 473–480.
- [11] G. E. Hinton, S. Osindero, and Y. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, Jul. 2006.
- [12] J.-D. Ren, H.-Y. Huang, and J. Bao, "The research on an algorithm for special two-hidden-layer artificial neural network," in *Proc. 2nd Int. Conf. Mach. Learn. Cybern.*, Xi'an, China, Nov. 2–5, 2003, pp. 1127–1131.
- [13] Md. M. Islam and K. Murase, "A new algorithm to design compact two-hidden-layer artificial neural networks," *Neural Netw.*, vol. 14, no. 9, pp. 1265–1278, Nov. 2001.
- [14] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Proc. Advances Neural Inf. Process. Syst.*, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1990, vol. 2, pp. 524–532.
- [15] J. de Villiers and E. Barnard, "Backpropagation neural nets with one and two hidden layers," *IEEE Trans. Neural Netw.*, vol. 4, no. 1, pp. 136–141, Jan. 1992.
- [16] S. Tamura and M. Tateishi, "Capabilities of a four-layered feedforward neural network: Four layers versus three," *IEEE Trans. Neural Netw.*, vol. 8, no. 2, pp. 251–255, Mar. 1997.
- [17] L. Ma and K. Khorasani, "Constructive feedforward neural networks using Hermite polynomial activation functions," *IEEE Trans. Neural Netw.*, vol. 16, no. 4, pp. 821–833, Jul. 2005.
- [18] R. Reed, "Pruning algorithms—A survey," *IEEE Trans. Neural Netw.*, vol. 4, no. 5, pp. 740–747, Sep. 1993.
- [19] P. Lauret, E. Fock, and T. A. Mara, "A node pruning algorithm based on a Fourier amplitude sensitivity test method," *IEEE Trans. Neural Netw.*, vol. 17, no. 2, pp. 273–293, Mar. 2006.
- [20] T. B. Ludermir, A. Yamazaki, and C. Zanchettin, "An optimization methodology for neural network weights and architectures," *IEEE Trans. Neural Netw.*, vol. 17, no. 6, pp. 1452–1459, Nov. 2006.
- [21] P. Guo, M. R. Lyu, and C. L. P. Chen, "Regularization parameter estimation for feedforward neural networks," *IEEE Trans. Syst., Man, Cybern. B: Cybern.*, vol. 33, no. 1, pp. 35–44, Feb. 2003.
- [22] C.-S. Leung, K.-W. Wong, P.-F. Sum, and L.-W. Chan, "A pruning method for the recursive least squares algorithm," *Neural Netw.*, vol. 14, no. 2, pp. 147–174, Mar. 2001.
- [23] M. Mezard and J.-P. Nadal, "Learning in feedforward layered networks: The tiling algorithm," *J. Phys. A: Math. Gen.*, vol. 22, no. 12, pp. 2191–2203, Jun. 1989.
- [24] R. E. Schapire, "The strength of weak learnability," *Mach. Learn.*, vol. 5, no. 2, pp. 197–227, Jun. 1990.
- [25] Md. M. Islam, X. Yao, S. M. S. Nirjon, M. A. Islam, and K. Murase, "Bagging and boosting negatively correlated neural networks," *IEEE Trans. Syst., Man, Cybern. B: Cybern.*, vol. 38, no. 3, pp. 771–784, Jun. 2008.
- [26] D. Parikh and R. Polikar, "An ensemble-based incremental learning approach to data fusion," *IEEE Trans. Syst., Man, Cybern. B: Cybern.*, vol. 37, no. 2, pp. 437–450, Apr. 2007.
- [27] R. Parekh, J. Yang, and V. Honavar, "Constructive neural-network learning algorithms for pattern classification," *IEEE Trans. Neural Netw.*, vol. 11, no. 2, pp. 436–451, Mar. 2000.
- [28] D. S. Phatak and I. Koren, "Connectivity and performance tradeoffs in the cascade correlation learning architecture," *IEEE Trans. Neural Netw.*, vol. 5, no. 6, pp. 930–935, Nov. 1994.
- [29] M. Lehtokangas, "Modified cascade-correlation learning for classification," *IEEE Trans. Neural Netw.*, vol. 11, no. 3, pp. 795–798, May 2000.
- [30] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Proc. 13th Int. Conf. Mach. Learn.*, 1996, pp. 148–156.
- [31] A. I. Rasiyah, R. Togneri, and Y. Attikiouzel, "Modeling 1-D signals using Hermite basis functions," *Proc. Inst. Elect. Eng.—Vis. Image Signal Process.*, vol. 144, no. 6, pp. 345–354, Dec. 1997.
- [32] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing*, vol. 1, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: MIT Press, 1986, pp. 318–362.
- [33] L. Prechelt, "PROBEN1—A set of neural network benchmark problems and benchmarking rules," Faculty Informat., Univ. Karlsruhe, Karlsruhe, Germany, Tech. Rep. 21/94, 1994.
- [34] A. F. Murray and P. J. Edwards, "Enhanced MLP performance and fault tolerance resulting from synaptic weight noise during training," *IEEE Trans. Neural Netw.*, vol. 5, no. 5, pp. 792–802, Sep. 1994.
- [35] S. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Upper Saddle River, NJ: Pearson Edu., 2004.
- [36] L. Prechelt, "Automatic early stopping using cross validation: Quantifying the criteria," *Neural Netw.*, vol. 11, no. 4, pp. 761–767, Jun. 1998.
- [37] T. Y. Kwok and D. Y. Yeung, "Objective functions for training new hidden units in constructive neural networks," *IEEE Trans. Neural Netw.*, vol. 8, no. 5, pp. 1131–1148, Sep. 1997.
- [38] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Netw.*, vol. 4, no. 2, pp. 251–257, 1991.
- [39] E. Kreyszig, *Introductory Functional Analysis With Applications*. New York: Wiley, 1989.
- [40] A. I. Galushkin, *Neural Networks Theory*. Berlin, Germany: Springer-Verlag, 2007, pp. 53–63.
- [41] L. Prechelt, "A quantitative study of experimental evaluation of neural network learning algorithms," *Neural Netw.*, vol. 9, no. 3, pp. 457–462, Apr. 1996.
- [42] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognit. Lett.*, vol. 27, no. 8, pp. 861–874, Jun. 2006.
- [43] R. Setiono and L. C. K. Hui, "Use of quasi-Newton method in a feedforward neural network construction algorithm," *IEEE Trans. Neural Netw.*, vol. 6, no. 1, pp. 273–277, Jan. 1995.
- [44] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.
- [45] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Comput. Oper. Res.*, vol. 13, no. 5, pp. 533–549, May 1986.

- [46] S. E. Fahlman, "An empirical study of learning speed in backpropagation networks," Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-CS-88-162, 1988.
- [47] Y. Le Cun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Proc. Advances Neural Inf. Process. Syst.*, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1990, vol. 2, pp. 598–605.
- [48] P. L. Bartlett, "The sample complexity of pattern classification with neural networks: The size of the weights is more important than the size of the network," *IEEE Trans. Inf. Theory*, vol. 44, no. 2, pp. 525–536, Mar. 1998.



Md. Monirul Islam received the B.E. degree from Khulna University of Engineering and Technology (KUET), Khulna, Bangladesh, in 1989, the M.E. degree from Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh, in 1996, and the Ph.D. degree from the University of Fukui, Fukui, Japan, in 2002.

From 1989 to 2002, he was a Lecturer and Assistant Professor with KUET. In 2003, he was an Assistant Professor of computer science and engineering with the Department of Computer Science and Engineering, BUET, where he is currently an Associate Professor. He is also a visiting Associate Professor, under the support of the Japanese Society for the Promotion of Sciences, with the Department of Human and Artificial Intelligence Systems, Graduate School of Engineering, University of Fukui. He has more than 80 refereed publications. His research interests include evolutionary robotics, evolutionary computation, neural networks, machine learning, pattern recognition, and data mining.

Dr. Islam won the First Prize in the Best Paper Award Competition at the Joint Third International Conference on Soft Computing and Intelligent Systems and the Seventh International Symposium on Advanced Intelligent Systems.



Md. Abdus Sattar received the B.Sc. degree in electrical and electronic engineering from the Engineering College, Rajshahi [(ECR); now the Rajshahi University of Engineering and Technology], Bangladesh, in 1975 and the M.Sc. degree in computer science and engineering from Bangladesh University of Engineering and Technology (BUET), Bangladesh, in 1990.

Starting his career as a Lecturer with ECR, he then moved to serve the industry as a Consultant Engineer abroad. In 1987, he returned to the academe as an Assistant Professor with the Department of Electrical and Electronic Engineering, Bangladesh Institute of Technology, Rajshahi. Since 1992, he has been an Assistant Professor with the Department of Computer Science and Engineering, BUET. His research interests include neural networks, bioinformatics, and pattern recognition, in particular implementing Bangla language in computer usage.



Md. Faijul Amin received the B.Sc. degree in computer science and engineering in 2004 from the Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, and the M.S. degree in human and artificial intelligence systems in 2009 from the University of Fukui, Fukui, Japan, where he is currently working toward the Ph.D. degree in systems design engineering.

He is also with the Khulna University of Engineering and Technology, Khulna, Bangladesh. His research interests include artificial neural networks in both real and complex domains.



Xin Yao (M'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, China, in 1985, and the Ph.D. degree, with a work on simulated annealing and evolutionary algorithms, from USTC in 1990. He received a Postdoctoral Fellowship in the Computer Sciences Laboratory, Australian National University, Canberra, Australia, in 1990 and continued his work on simulated annealing and evolutionary algorithms.

From 1985 to 1990, he was an Associate Lecturer and a Lecturer with USTC. In 1991, he joined the Knowledge-Based Systems Group, Commonwealth Scientific and Industrial Research Organization, Division of Building, Construction and Engineering, Melbourne, Australia, 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, University of New South Wales, Australian Defence Force Academy, where he was later promoted as a Senior Lecturer and an Associate Professor. Attracted by the English weather, he moved to the University of Birmingham, Birmingham, U.K., as a Professor of computer science on April 1, 1999. He is currently the Director of the Centre of Excellence for Research in Computational Intelligence and Applications, School of Computer Science, University of Birmingham, and the Changjiang Visiting Chair Professor (Cheung Kong Scholar) with USTC. He is an Associate Editor or a Member of the Editorial Board of 12 journals and the Editor of the *World Scientific Book Series on Advances in Natural Computation*. He has more than 300 refereed publications. His research interests include neural network ensembles, evolutionary computation, and real-world applications.

Dr. Yao received the 2001 IEEE Donald G. Fink Prize Paper Award and several other best paper awards. He is a Distinguished Lecturer of the IEEE Computational Intelligence Society. From 2003 to 2008, he was the Editor-in-Chief of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION. He has given more than 50 invited keynote and plenary speeches at conferences and workshops worldwide.



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.

In 1984, he joined the Department of Information Science, Toyohashi University of Technology, Toyohashi, Japan, as a Research Associate. In 1988, he was an Associate Professor with the Department of Information Science, Fukui University, Fukui, Japan, where he became a Professor in 1992. Since 1999, he has been a Professor with the Department of Human and Artificial Intelligence Systems, Graduate School of Engineering, University of Fukui, Fukui, Japan, where he is also with the Research and Education Program for Life Science.

Dr. Murase is a member of the Institute of Electronics, Information, and Communication Engineers, the Japanese Society for Medical and Biological Engineering, the Japan Neuroscience Society, the International Neural Network Society, and the Society for Neuroscience. He is a Member of the Board of Directors of Japan Neural Network Society, a Councilor of the Physiological Society of Japan, and a Councilor of the Japanese Association for the Study of Pain.