Performance and Efficiency: Recent Advances in Supervised Learning

SHENG MA AND CHUANYI JI

This paper reviews recent advances in supervised learning with a focus on two most important issues: performance and efficiency. Performance addresses the generalization capability of a learning machine on randomly chosen samples that are not included in a training set. Efficiency deals with the complexity of a learning machine in both space and time. As these two issues are general to various learning machines and learning approaches, we focus on a special type of adaptive learning systems with a neural architecture. We discuss four types of learning approaches: training an individual model; combinations of several well-trained models; combinations of many weak models; and evolutionary computation of models. We explore advantages and weaknesses of each approach and their interrelations, and we pose open questions for possible future research.

Keywords— *Evolutionary computation, hybrid models, neural networks, supervised learning.*

I. INTRODUCTION

In many important application areas such as signal processing, pattern recognition, control, and communication, nonlinear adaptive systems are needed to approximate underlying nonlinear mappings through learning from examples. In order for approximations to be sufficiently accurate, a good performance is required for nonlinear adaptive systems. Meanwhile, many applications, especially those in emerging areas of wireless communication and networking [24], [38], [79], [95], require the learning to be done in real time in order to adapt to a rapidly changing stochastic environment. Other applications such as data mining and searching the Web need to deal with very large data sets [37], [66], and thus the learning time must scale nicely with respect to the size of data sets. Since the size of learning machines determines the memory required for implementation, a learning machine with a compact structure is preferred. Therefore, a challenging problem is how to develop adaptive learning systems with a compact structure that can achieve good performance and be adapted in real time. The goal of this paper is to address the important issues of performance and real-time learning for nonlinear adaptive learning machines by reviewing recent work in the interdisciplinary areas of adaptive learning systems, statistics, and information theory.

Manuscript received October 30, 1998; revised April 2, 1999. This work was supported by the National Science Foundation under ECS-9312594 and (CAREER) IRI-95025 18.

S. Ma is with IBM T. J. Watson Research Center, Hawthorne, NY 10532 USA (e-mail: shengma@us.ibm.com).

C. Ji is with the Department of Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY 12180 USA (e-mail: chuanyi@ecse.rpi.edu).

Publisher Item Identifier S 0018-9219(99)06910-8.

There are two aspects when these issues are investigated: architecture of adaptive learning machines and learning scenarios (approaches). The learning scenario considered is the supervised learning for nonparametric nonlinear regression including classification as a special case. We discuss several general frameworks, such as the expectation-maximization (EM) framework, the combination scheme, weak learning, and evolutionary algorithms, all of which aim at improving the efficiency and performance of a learning machine. In order to be comprehensive, a neural network is used as a sample architecture to show how these general frameworks are applied. The reason why neural networks are chosen is that they have been shown to be universal approximators to a general class of nonlinear functions [9] and have become popular recently. The general framework can be applied to other learning machines. This, however, is not the focus here.

In supervised learning, performance addresses the problem of how to develop a learning machine to achieve optimal performance on samples that are not included in a training set. Efficiency deals with the complexity of a learning machine in both space and training time. Specifically, the space complexity of a neural network refers to its size, and the time complexity characterizes the computational time needed to develop such a neural network. These three issues are interrelated.

The performance of a supervised learning system is characterized by its generalization error, which measures the distance between the output function of a trained model and an underlying target function. Most existing methods for training neural networks in supervised learning suffer from an intrinsic problem in pattern recognition: the bias and variance dilemma [39], [47]. That is, if a neural network is too large,[1] it may overfit a particular training set and

[1] For instance, feedforward neural networks with too many neurons.

thereby fail to maintain good generalization error. A small neural network, however, may be sufficient to approximate an optimal solution. In addition, one important algorithmic problem is how to deal with a complex optimization problem with possibly many local minima.

The size of a learning machine can be characterized by the space complexity, which is related to the number of free parameters (for instance, the number of weights of a neural network). A learning machine is considered to be efficient in space if its space-complexity scales as a polynomial function in terms of the dimension of feature vectors.[2] It has been found that compact adaptive learning systems like multilayer feedforward neural networks are efficient approximators of a wide class of smooth functions. They possess a polynomial space complexity [10]. Other learning machines, which consist of localized models such as nearest neighbor classifiers [27], Parzen windows [35], and linear combinations of localized basis functions [10], [80], may suffer from the so-called "curse of dimensionality." That is, their space-complexity scales exponentially with the dimension of the feature vectors [10], [65], [85].

Learning time can be characterized by the time complexity as a scaling property with respect to the dimension of feature vectors. An adaptive learning system is considered efficient in time if the time complexity is polynomial. Training (nonlinear) feedforward neural networks and classifiers is slow, and in general, NP-complete [18], [59]. But training the aforementioned localized learning machines can be done quickly. For example, nearest-neighbor classifiers simply remember all training samples and therefore do not require training. This presents a dilemma that exists between performance, space complexity, and time complexity. Therefore, an important issue is whether or not it is possible and how to develop a neural network that can learn in real time while achieving a desired performance with a polynomial space complexity.

In this work, we address these three fundamental issues: performance; efficiency; and space-complexity. We discuss how they motivate the search for new solutions for adaptive learning machines with a neural architecture. To achieve our goal within a limited scope, we focus on reviewing recent work in three areas: training an individual neural network; combinations of well-trained models; and combinations of weak models. We also discuss hybrid methods, such as evolutionary computation in the emerging area of soft computing, and their role in tackling these fundamental issues.

This paper is organized as follows. Section II gives the background knowledge on performance and efficiency. Section III provides an overview to the approaches to be discussed in this paper. Section IV reviews neural networks that utilize a fixed structure to tackle the issues of performance and efficiency. As finding an appropriate structure is very difficult, approaches have been developed to combine models. Section V reviews the research on combining well-trained models to improve the performance. Section VI then discusses combinations of weak classifiers and how combined weak classifiers make a tradeoff among performance, time complexity and space complexity. Section VII introduces hybrid methods, such as evolutionary computation, and their role in tackling issues of performance and efficiency. We conclude the paper in Section VIII. As there is a large volume of related work, our selections of material may be subjective, and we apologize for any significant omission.

II. Background: Performance and Efficiency

A. Notation

In a supervised learning environment, let $D = \{\mathbf{x}_n, t_n\}_{n=1}^{N}$ denote N training samples pairs, where $\mathbf{x}_n \in R^d$ is a d-dimensional feature vector of the nth sample and t_n is the corresponding target. For simplicity, t_n is assumed to be a scalar in this paper. Furthermore, we assume that there is a function $f(\cdot)$ so that $t_n = f(\mathbf{x}_n)$.[3] For the function approximation problem, t_n is a real number. For classification, t_n indicates the class to which the nth sample belongs.

Neural networks are a class of nonlinear models that consist of interconnected nonlinear processing nodes. As an example, a two-layer feedforward network with one linear output unit, d input units, and L sigmoidal hidden units is shown in Fig. 1. Let $\mathbf{W}^{(1)}$ be a weight matrix at the first layer and its jth column $\mathbf{w}_j^{(1)}$ ($1 \le j \le L$) be the weight vector connecting the jth hidden unit to the inputs. Let $\mathbf{w}^{(2)}$ be a weight vector at the second layer and its jth element $w_j^{(2)}$ denote the weight connecting the jth hidden unit to the output. $\mathbf{h}_n$ is a vector that represents the outputs of hidden units corresponding to input $\mathbf{x}_n$. The jth element $h_{j,n}$ ($1 \le j \le L$) of $\mathbf{h}_n$ is equal to

$$h_{j,n} = g\left(\mathbf{w}_j^{(1)T}\mathbf{x}_n\right) \tag{1}$$

where $g(\cdot)$ is the sigmoidal transfer function. The output of the network, y_n, corresponding to an input $\mathbf{x}_n$ can be expressed as

$$y_n = \mathbf{w}^{(2)T}\mathbf{h}_n. \tag{2}$$

B. Performance and Efficiency

1) Performance: Let $f_D(\mathbf{x};\mathbf{w})$ be a model with a set of parameters $\mathbf{w}$ and trained on training set D. The performance of $f_D(\mathbf{x};\mathbf{w})$ can be measured in terms of the difference between a function $f(\mathbf{x})$ to be approximated and its approximation $f_D(\mathbf{x};\mathbf{w})$ through the squared norm

$$\int |f(\mathbf{x}) - f_D(\mathbf{x};\mathbf{w})|^2 p(\mathbf{x})\, d\mathbf{x} \tag{3}$$

where $p(\mathbf{x})$ is the probability density function of $\mathbf{x}$. If training samples are drawn randomly, the expected squared

[2]Here the dimension of feature vectors is used as a measure of the complexity of a problem.

[3]For simplicity, this paper does not address possible noise terms.

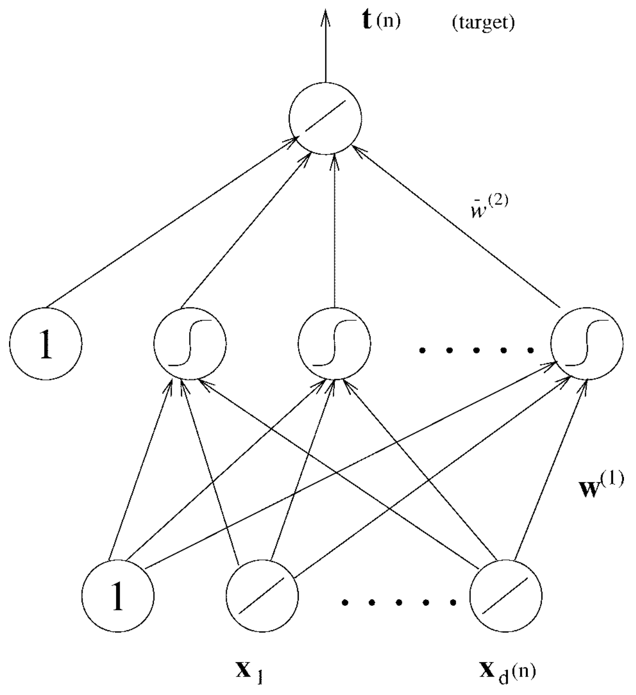

Fig. 1. The structure of a two-layer feedforward network.

norm is

$$\mathbf{E}_D(\|f(\mathbf{x}) - f_D(\mathbf{x};\mathbf{w})\|^2) = \mathbf{E}_D\left(\int |f(\mathbf{x}) - f_D(\mathbf{x};\mathbf{w})|^2 p(\mathbf{x})\, d\mathbf{x}\right) \quad (4)$$

where the expectation ($\mathbf{E}$) is over all possible training sets D of the same size. The quantity $\mathbf{E}_D(\|f(\mathbf{x}) - f_D(\mathbf{x};\mathbf{w})\|^2)$ can be considered as a measure of the performance. It is also called the generalization error of $f_D(\mathbf{x};\mathbf{w})$, since it measures the average performance of $f_D(\mathbf{x};\mathbf{w})$ in approximating the unknown function $f(\mathbf{x})$ rather than fitting N training samples alone.

When pattern classification is considered as a special case of nonlinear regression, generalization error can be defined as the probability of incorrect classification, which is $\mathbf{E}_D(\Pr(f_D(\mathbf{x};\mathbf{w}) \neq t))$.

2) Efficiency: The space complexity S of $f_D(\mathbf{x};\mathbf{w})$ is the number of free parameters l of $f_D(\mathbf{x};\mathbf{w})$ when the generalization error is no bigger than $\epsilon_g > 0$. For example, if a two-layer feedforward neural network is considered, l can be either the number of hidden units or the total number of independent weights of the network. The time-complexity of an adaptive learning system is defined as the expected training time needed to obtain a learning system when the generalization error is bounded by ϵ_g.

The scaling property of the time complexity and space complexity defines the efficiency of an adaptive learning machine (or algorithm) whose generalization error is no bigger than a given quantity $\epsilon_g > 0$. $f_D(\mathbf{x};\mathbf{w})$ is said to be efficient in space if S scales polynomially in terms of the dimension d of feature vectors, ϵ_g, and other related parameters. If the time complexity scales polynomially in term of these quantities, the learning machine is said to be efficient in time. If the efficiency can be achieved both in space and time, the learning machine is said to be efficient.

3) Relationships Between Performance and Efficiency: Performance and efficiency are interrelated. This can be understood through an example of a two-layer feedforward neural network.

Suppose a training algorithm exists that can obtain the weights of each hidden unit in polynomial time. The algorithm is not efficient if the number of hidden units needed is exponential in terms of the dimension of feature vectors. This is because the total training time can be estimated as the time needed to train a single hidden unit multiplied by the number of hidden units in the networks. In other words, a polynomial space complexity is needed to achieve the polynomial time complexity for training the entire network for this example. Together, space efficiency and time efficiency can define the efficiency of a learning algorithm for two-layer feedforward neural networks. This idea will be further explained in Section VI in the context of combinations of weak classifiers.

Efficiency has to be defined together with performance, i.e., the generalization error of a resulting two-layer network should be smaller than a desired quantity[4] when the space complexity and time complexity are examined. This is because the concept of efficiency would be meaningless without a requirement on generalization performance. For example, a binary classifier with a generalization error of $1/2$ can certainly be obtained using an efficient (yet trivial) algorithm that only performs random guessing.

C. Performance Issues

There are two important factors affecting the performance: the bias and variance dilemma, and possibly a complex objective function with many local minima.

The first issue is intrinsic and is independent of algorithms used. The second issue is algorithm and problem dependent. In the following, we further discuss the first issue and its relationship with the second issue.

1) Bias and Variance Dilemma: The generalization error is affected by two factors: bias and variance. To define the bias and variance, let $f(\mathbf{x};\hat{\mathbf{w}})$ be the best model in the model space, that is, $\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \int_{\mathbf{x}}(f(\mathbf{x}) - f(\mathbf{x};\mathbf{w}))^2 p(\mathbf{x})\, d\mathbf{x}$. Therefore, the solution $\hat{\mathbf{w}}$ does not depend on the training data. Furthermore, the bias $\mathrm{Bias}(\mathbf{w})$ and variance $\mathrm{Var}(\mathbf{w})$ for a model can be defined as

$$\mathrm{Bias}(\mathbf{w}) = \mathbf{E}(f(\mathbf{x}) - f(\mathbf{x};\hat{\mathbf{w}}))^2 \quad (5)$$

where the expectation is on a randomly drawn $\mathbf{x}$, and

$$\mathrm{Var}(\mathbf{w}) = \mathbf{E}_D\big(\mathbf{E}\big(f_D(\mathbf{x};\mathbf{w}) - f(\mathbf{x};\hat{\mathbf{w}})\big)^2\big). \quad (6)$$

The inner expectation is on $\mathbf{x}$ and the outer one is on randomly drawn training sets of the same size. Therefore, the generalization error can be decomposed into, and bounded by, a sum of the bias and variance [10], [22], [39]

$$\mathbf{E}_D\big(\|f(\mathbf{x}) - f_D(\mathbf{x};\mathbf{w})\|^2\big) \le 2(\mathrm{Bias}(\mathbf{w}) + \mathrm{Var}(\mathbf{w})). \quad (7)$$

[4]This is no smaller than the Bayes error of a given problem [35]. One could assume the Bayes error is small.

The first term of the right-hand side is the error due to the bias that can be caused by an inappropriate choice of the size of a class of models when the number of training samples is assumed to be infinite. The latter is the error corresponding to the variance and is caused by the finite number of training samples.[5]

In a special case when a class of models was chosen to be two-layer feedforward neural networks with L sigmoid hidden units,[6] the bias and variance were bounded explicitly by Barron [10]. In particular, $O(1/L)$ and $O((Ld\log N)/N)$ are upper bounds for the bias and variance respectively, where $O(z)$ represents a quantity in the order of z. The fact that $O(1/L)$ decreases with the number of hidden units suggests that a large neural network with many free parameters is less biased. Since it has been shown by Barron [10] that feedforward neural networks are capable of approximating a wide class of smooth functions when the number of hidden units is sufficiently large, a larger set of neural networks (with a larger L) certainly contains a better approximation of $f(\mathbf{x})$ than a smaller set of neural networks (with L being smaller). That is why the bias is smaller when L is larger. However, when the number of training samples is finite, a network with an excessively large space complexity will overfit the training set. This can be understood from the concept of information capacity of neural networks [1], [13], [28], [52], [74] in information theory. The information capacity is the maximum number of training samples that can be memorized by a neural network given a certain space complexity. When the space complexity exceeds the information capacity, learning machines are so large that they only memorize training samples. Since there are many such neural networks with the same space complexity but different choices of weights which can memorize training samples, their (generalization) performance varies and thus leads to a large variance. That is, the average performance can decrease as L gets larger. This can be observed from the term $O((Ld\log N)/N)$, which increases with respect to L. Therefore, a tradeoff needs to be made between bias and variance.

2) Bias Variance and Local Minima: An issue that needs to be clarified is whether the bias and variance are related to local minima, or the effectiveness of an algorithm at finding a good solution. In our view, the bias by definition only depends on the structure of a class of neural networks but not the choice of values of weights and is therefore independent of any training algorithm. To understand the variance as defined, we can consider the sample variance. For any given (randomly drawn) training set D of size N, a network corresponding to $f_D(\mathbf{x};\mathbf{w})$ can be found, which contributes to one sample in the sample variance. The true variance can be estimated through a sufficiently large number of such samples. In the definition given by Barron [10], it was assumed that each $f_D(\mathbf{x};\mathbf{w})$ corresponded to the global minimum of the error function, i.e., it was the best network that can be obtained given information from a training set. To define the variance depending on a specific algorithm and applications, the theoretical result may lose its generality. This definition (given by Barron) simply suggests that the bias and variance dilemma is intrinsic for nonparametric regression (even if one had the most powerful algorithm to obtain the globally optimal network). Therefore, such a definition on the variance can also be considered to be algorithm independent.

In practice, for a given training set and the chosen error function, different algorithms result in different networks corresponding to different $f_D(\mathbf{x};\mathbf{w})$'s. If an algorithm (say, an evolutionary algorithm to be introduced later) is always better at finding a globally optimal solution, and another algorithm (say, a gradient-descent algorithm) gets stuck at a local minimum more often, then the former would have a smaller sample variance than that of the latter.

III. Overview of Approaches

Before we go into details, a quick overview on what we will soon discuss may elucidate the logical organization of this paper.

Focusing on performance and efficiency, we start our review from an individual model. For training an individual model, performance can be improved by balancing bias and variance through either finding a model with optimal size or adding a regulation term in an objective function to confine the complexity of a model. Efficiency in finding a model can be achieved through fast training algorithms based on the "divide-and-conquer" principle.

However, finding an optimal size is usually difficult if not impossible. To address the performance issue, a combination of well-trained models has been proposed. The essential idea is to pick a slightly oversized model as a base for combination. Since the base model is oversized, it has a low bias but a high variance. The algorithm, therefore, relies on combinations to diminish the overall variance, and thus the generalization error.

Although the combination of well-trained models relaxes the need to optimize the size of a model, its learning time increases with the number of models to be combined. The combination of weak models aims at improving the efficiency of the combination of well-trained models while preserving the nice performance property. In particular, this approach uses the similar combination scheme but it chooses a weak model as a base model. A weak model is a model whose performance is only slightly better than random guessing and thus has a large bias but a small variance. From the performance perspective, since different weak models are forced to learn different parts of a problem, combinations of many such weak models can diminish the overall bias and thus achieve good generalization. From the time-efficiency point of view, since the base model performs only slightly better than random guessing, it can be obtained efficiently. Furthermore, this approach uses an incremental learning procedure. The whole training process can be very efficient.

Since there is still a large body of related work in these three areas, we further narrow our scope in each area.

[5] This is equivalent to the space complexity versus the number of training samples.

[6] The number of free parameters is thus approximately $L(d+1)$.

Specifically, for training an individual model, we focus on the issue of the time efficiency and illustrate how the "divide-and-conquer" principle is applied to the reviewed algorithms. For combinations of well-trained models, since there are papers with an extensive review on this subject [33], [77], we focus on one class of algorithms that hopefully serves as a bridge to the next topic on combinations of weak classifiers. More thorough review is given on combinations of weak models both on algorithms used and theoretical results.

As will soon be discussed, randomized algorithms play a key role in combinations of both well-trained and weak models. They are also one of a few approaches that can hopefully result in polynomial-time algorithms and achieve global optimization. This leads us to further discussing hybrid methods such as evolutionary computation in the emerging area of soft computing and illustrating the relationships with combinations of weak classifiers.

IV. Performance and Efficiency: Training an Individual Model

A. Performance

In order to make a tradeoff between bias and variance when a single network is used, we need to find the optimal architecture of a neural network. This is equivalent to determining an optimal space complexity, for example, the optimal number of hidden units and layers for multilayer neural networks. Tremendous efforts have been made on estimating and finding the optimal architecture of a learning system using a finite number of training samples. These approaches include computational learning theory [13], [39], [49], [53]–[55], [96]–[98], various statistical methods including cross validation and model selection [9]–[11], [81], [83], [100], [104], and stochastic exploration of an optimal structure based on evolutionary algorithms [73], [105]. Due to inherent difficulties in accurately estimating the (space) complexity of a nonlinear adaptive system like a neural network with limited training samples, finding the optimal architecture for a learning machine remains a very difficult task.

Another practical difficulty of training a neural network is that the error function (the objective function) to be minimized is usually very complex with many local minima and possibly flat areas [40]. Therefore, a deterministic gradient-based approach may be trapped by a local minimum rather than a global minimum. To deal with this issue, stochastic algorithms have been developed. Since stochastic algorithms are also used in our next two subjects as a general framework for randomization, we will leave the further discussion on these algorithms to Section VII.

B. Space Complexity

Although an exact optimal architecture is difficult to obtain, nonlinear adaptive systems such as multilayer feedforward neural networks have been shown to be efficient in space. In particular, Barron [10] showed that the space complexity of two-layer feedforward networks with sigmoidal hidden units is polynomial when used to approximate a wide class of smooth functions. The space complexity of linear combinations of localized basis functions, however, is exponential when used to approximate the same class of functions. Although feedforward two-layer neural networks can also be regarded as linear combinations of (sigmoidal) basis functions, they are efficient in space because of two major reasons: 1) sigmoidal functions are nonlocal and 2) sigmoidal functions at the hidden layer when regarded as basis functions are parameterized by their weights. Those weights effectively adapt the sigmoidal (basis) functions to the feature space and thus result in a good scaling property on the space complexity with respect to the dimension of feature vectors.

C. Fast Training Algorithms

Finding a nonlinear adaptive system like a neural network with an appropriate structure is a nonlinear optimization problem, which can be complex and slow. One of the most widely used algorithms for training neural networks is backpropagation (BP) based on gradient descent [90], [103]. Although this simple approach has been successfully used to train neural networks for a wide range of applications, the algorithm can be notoriously slow when used to train a large network in solving a complex problem. This is because neural networks with a complicated structure result in complex error functions that are difficult to minimize.

In fact, it has been shown that training a multilayer neural network of a fixed structure is NP complete [18], [59]. It is not clear whether an approximation of an optimal solution can be found in polynomial training time either. Therefore, to tackle the issue of time complexity, efforts have been focused mostly on developing "fast" training algorithms that can improve upon the training time compared with, for instance, that of BP. There are two categories of fast training algorithms. The first one is motivated by the "divide-and-conquer" principle and tries to decompose the complicated problem into a set of relatively easy subproblems. The second class of algorithms is motivated by stochastic algorithms and tries to "escape" from local minima through randomized approaches. We discuss algorithms in the first category next and the second category in Section VII.

Three major types of fast training algorithms developed in the first category are: 1) decision-tree algorithms; 2) incremental learning algorithms for training multilayer neural networks; and 3) EM algorithms. Although they are in different forms, they were all developed based on the "divide-and-conquer" principle.

1) Decision Tree: A decision tree is a directed, acyclic graph in which each node is either a decision node with two or more successors or a leaf node. Every leaf is labeled as a class, while a decision node has a model for deciding to which successor a sample (feature vector) belongs. Usually, a model at a decision node can be as simple as a linear model (or a so-called perceptron) or just a threshold on an attribute. A decision tree can be considered as a special case of multilayer feedforward neural networks [4].

The decision tree algorithm can be considered as a typical example of "divide-and-conquer." It is a recursive algorithm (see [19] and [86] for more elaborate review). The algorithm starts with a root node which utilizes all training data. First, a best model is computed according to a criterion for best separating the training data into two sets. Then the tree grows to the next level with two children so that the left child only takes care of one part of the training data, and the right child looks at the rest. The left and the right children have their own children in a similar fashion, and the tree grows until a stopping criterion is satisfied.

The decision tree algorithm is very efficient for two reasons. First, each node usually uses a simple decision model, (e.g., a perceptron or a threshold on one attribute). Secondly and more importantly, a complicated problem is divided by a decision node so that its successors only focus on parts of the problem instead of the entire problem.

2) Incremental Learning: Incremental learning trains one base model at a time using the residual error from the previously trained models. Once a base model is trained, it is fixed and combined with all previously trained models. The new residual error is calculated for training the next base model until a certain stopping criterion is met. Through this algorithm, training a large complicated model is reduced to training an individual base model sequentially. As an example, to train a two-layer feedforward neural network [36], a neuron is treated as a base model. Since incremental algorithms reduce the task of training an entire neural network into training individual neurons sequentially, they have been shown to be one or two orders of magnitude faster than gradient-descent-based algorithms [23], [36], [46].

3) EM Algorithms: As the previous two algorithms were developed based on heuristics, fast training algorithms based on EM algorithms were formulated through a better established theoretical framework [58], [69]–[71]. The EM algorithm was originated from statistics [31] for finding the maximum likelihood. It was introduced into the learning community in the past few years to speed up training. Specifically, EM algorithms were derived for training the stochastic Boltzmann machines [2], [25], the mixture of experts [16], [58], two-layer feedforward neural networks [71], and fully connected recurrent neural networks [69], [70].

In the following, we first review the general EM framework [31]. We discuss how to apply the EM framework in a supervised learning environment [58]. We then provide an EM algorithm for training a two-layer neural network [71].

a) EM framework: The EM algorithm can be viewed as a statistical framework for maximum likelihood estimation [31]. Let D_I be a set of data observed directly, and let Θ be a set of parameters characterizing the corresponding distribution of the random variables. The maximum likelihood problem is to find an optimal set of parameters Θ^{opt} through maximizing the log likelihood of the data. That is

$$\Theta^{\text{opt}} = \arg\max_{\Theta} \ln P(D_I|\Theta). \tag{8}$$

Although the above optimization problem can be done through a gradient-based approach, it can be computationally difficult to maximize the original log likelihood function directly. How can such an optimization problem be simplified? Intuitively, a complicated problem may be decomposed into simpler problems through carefully chosen auxiliary variables. These auxiliary variables may not be observed directly but may be estimated. These variables are called missing variables and denoted as D_{mis}. The EM algorithm provides a framework to utilize such missing variables and simplify a complex optimization problem into several much simpler problems. The way to do so is through iterating between a so-called expectation(E)-step, and a maximization(M)-step. The E-step estimates the missing variables, and the M-step uses the estimated variables to find a locally optimal solution.

Specifically, in the E-step, the following conditional expectation can be computed (with respect to hidden variables) of the complete data (D_c) log likelihood given incomplete data D_I and the previous parameters $\Theta^{(p)}$ at the pth step:

$$Q\Big(\Theta|\Theta^{(p)}\Big) = \mathbf{E}\Big\{\ln P(D_c|\Theta)|D_I,\, \Theta^{(p)}\Big\} \tag{9}$$

where $D_c = \{D_I,\, D_{\text{mis}}\}$. D_c is called complete data including both missing data and incomplete data. The notation $Q(\Theta|\Theta^{(p)})$ indicates that the conditional expectation is a function of Θ and the expectation is evaluated using $\Theta^{(p)}$ as parameters in the probability density function. $P(\cdot)$ represents a probability density function. $\mathbf{E}$ stands for the expectation operation. In the M step, new parameters $\Theta^{(p+1)}$ are found by maximizing the expected log likelihood $Q(\Theta|\Theta^{(p)})$. The algorithm alternates between the E and M step and terminates when a certain convergence criterion is satisfied. Local convergence of the EM algorithm is guaranteed. That is, after every (E-M) iteration, the original log likelihood does not decrease, i.e., $\ln P(D_I|\Theta^{(p)}) \le \ln P(D_I|\Theta^{(p+1)})$. As a result, the optimal set of parameters Θ^{opt} can be obtained. In practice, this implies convergence to a local maximum of the original log likelihood $\ln P(D_I|\Theta)$.

b) EM for supervised learning: Jordan *et al.* [58] introduced the EM framework into the supervised learning environment. Specifically, the observed data, i.e., incomplete data, can be defined as $D_I = \{\mathbf{x}_n,\, t_n\}_{n=1}^N$ [58]. Moreover, if a set of hidden targets denoted as $\{\mathbf{z}_n\}_{n=1}^N$ can be chosen, the complete data corresponds to $D_c = \{\mathbf{x}_n,\, t_n,\, \mathbf{z}_n\}_{n=1}^N$. For notational simplicity, we will use $\{\cdot\}$ to represent $\{\cdot\}_{n=1}^N$. Therefore, the Q function (9) can be rewritten as the following, when training samples are assumed independently drawn from the input domain:

$$Q\Big(\Theta|\Theta^{(p)}\Big) = \ln P(\{\mathbf{x}\}|\Theta) + \int P\Big(\{\mathbf{z}\}|\{t\},\, \{\mathbf{x}\},\, \Theta^{(p)}\Big) \cdot \ln P(\{\mathbf{z}\},\, \{t\}|\{\mathbf{x}\},\, \Theta)\, d\{\mathbf{z}\}. \tag{10}$$

Therefore, to apply the EM algorithm, we need to define a proper set of hidden targets $\{\mathbf{z}\}$ and the probability models of $P(\{\mathbf{z}\}|\{t\}, \{\mathbf{x}\}, \Theta)$ and $P(\{\mathbf{z}\}, \{t\}|\{\mathbf{x}\}, \Theta)$.

c) EM for neural networks: To show a relatively simple example, we apply the EM framework to training a two-layer feedforward neural network. Our discussion is based mostly on [70]. More detailed treatment for more complicated (recurrent) neural networks can be found in [69] and [71].

In [69] and [71], the hidden variables have been chosen to be the desired targets at the hidden layer denoted by $\mathbf{z}_n$ for the nth training sample. This is motivated by the fact [48], [64], [89], [91] that if the targets (missing variables) of each individual neuron could be obtained in addition to the input–output training examples, training a complicated neural network with many interconnected hidden neurons could be reduced to training a sequence of individual neurons.

To establish probability models, Gaussian models have been established in [69] and [71] based on previous work [67], [72], [104]. Through some tedious algebraic manipulations, the following algorithm has been derived [71].

1) $p = 0$ and initialize the weights at both layers ($\mathbf{W}^{(1)}$ and $\mathbf{w}^{(2)}$) randomly.
2) E-step: Compute the expected hidden targets for the hidden units $\hat{\mathbf{z}}_n$ as follows:

$$\hat{z}_{j,n} = h_{j,n}^{(p)} + \frac{\lambda_2 w_j^{(2)^{(p)}}}{\lambda_1 + \lambda_2 \|\mathbf{w}^{(2)^p}\|^2} e_n \tag{11}$$

where λ_1 and λ_2 are weighting parameters. $h_{j,n}^{(p)}$ represents the jth element of $\mathbf{h}_n$ (given in (1) evaluated at the pth step. $e_n = t_n - y_n^{(p)}$ is the training error of the nth training sample at the pth step.
3) M-step: Compute $\mathbf{W}^{(1)}$ and $\mathbf{w}^{(2)}$ as follows

$$\mathbf{w}_j^{(1)^{(p+1)}} = \arg\min_{w_j^{(1)}} \sum_n \left(\hat{z}_{jn} - g(\mathbf{w}_j^{(1)^T} \mathbf{x}_n)\right)^2 \tag{12}$$

where $\mathbf{w}_j^{(1)^{(p+1)}}$ stands for the new weight vector at the first layer, for $1 \leq j \leq L$. This step is equivalent to training individual sigmoidal (hidden) units in parallel

$$\mathbf{w}^{(2)^{(p+1)}} = \arg\min_{\mathbf{w}^{(2)}} \sum_n \left(\mathbf{w}^{(2)^T} \hat{\mathbf{z}}_n - t_n\right)^2 \tag{13}$$

where $w^{(2)^{(p+1)}}$ are the new second-layer weights. This can be accomplished by training individual single neurons simultaneously.
4) Let $p = p + 1$, and go back to step 2) until certain convergence conditions are satisfied.

The main idea of the derived EM algorithm for training a two-layer neural network can be summarized as follows. At each iteration, the expected hidden targets are evaluated first through the E-step (11). The expected hidden targets then serve as the targets at the hidden layer to train the first-layer weights $\mathbf{W}^{(1)}$ (12). Since the hidden targets for the hidden units are completely separable, finding the first-layer weights can be done through training M individual neurons simultaneously. Furthermore, the hidden targets also serve as inputs to the second layer, and the quadratic training error between the outputs and the desired targets is minimized to obtain the new second-layer weights $\mathbf{w}^{(2)}$ (13). This is equivalent to training a single linear neuron. Therefore, training the original two-layer (nonlinear) network is decomposed into training a set of single neurons, which can be done much faster than training the original (nonlinear) layer network. The derived EM algorithms were tested extensively on various applications [70], [71] and were shown to reduce the training time by a factor of 10–20 as compared with BP.

4) Discussion: Three types of fast training algorithms have been reviewed in this section that apply the “divide-and-conquer” principle to improve training time.

Decision tree algorithms divide a problem (or a training set) recursively by the tree structure so that nodes (or base models) in the tree deal with successively easier problems moving down the tree. Therefore, a decision tree algorithm can be viewed as implementing “divide and conquer” in a “vertical” fashion. The incremental algorithms train a base model individually using residual error. Therefore, this scheme can be viewed as implementing the “divide and conquer” in a “horizontal” fashion. The EM algorithm works in a slightly different way. It divides a problem into small pieces iteratively using the “divide-and-conquer” procedure so that each base model can be trained in the maximization step. Once training is completed, it redivides the problem and reestimates information needed to train each small piece in the expectation step. Each piece is then trained again in the M step. The process continues until a convergence criterion is met.

D. Open Problems

Although fast training algorithms have been shown to improve the training time substantially, the improvement is usually measured heuristically. The theory on training time is still missing.

Many algorithms have been proposed to adapt the structure of a single model in order to find the optimal trade-off between the bias variance. However, these algorithms are mostly developed through heuristics and lack a solid theoretical ground. The algorithmic complexity of these algorithms may also be higher. It has not been proved theoretically whether and how to develop an algorithm that is efficient for achieving good performance through adapting its structure. As a result, achieving a good generalization performance of an individual model is still considered to be a challenging open question.

V. Performance: Combinations of Well-Trained Models

To alleviate the difficulties in finding an optimal structure of a single nonlinear adaptive system (neural network), methods are proposed to combine different models. The main idea is to train multiple models, such as neural networks, decision trees, or classifiers individually, and then combine them as an ultimate model. The hope for using a combination to improve the generalization performance is that if individual models make mistakes differently, a combined model would be able to improve upon the performance of a single model.

Tremendous efforts have been made to investigate whether a combined model can indeed improve the performance, and how to combine models to achieve good performance [3], [88]. Specifically, combinations of experts have shown to be able to work at least as well as the best expert in the pool of experts on predicting binary strings [26], [68]. Combinations of neural networks [84], [102] and regressors [21] have also been used for both classification and function approximation problems. Somewhat similar ideas were also used for collective agents such as in classifier systems and genetic algorithms [43], [50], [51].

As a good review on this subject was offered by Dietterich [33], in this section we do not intend to survey all aspects of combinations of well-trained models. Rather, we focus on the general issues that are relevant to the next subject we will discuss on combinations of weak classifiers.

A. Key Issues

A combined model can be represented in a simple form. Let $h_k(\mathbf{x})$'s be a set of total K models defined on a feature vector $\mathbf{x} \in \mathbf{R}^d$, where a $h_k(\mathbf{x})$ can be a neural network (or a single neuron). Let w_k be the weighting factor for the kth model. The combination of these K models can be represented as

$$f_K(\mathbf{x}) = \sum_{k=1}^{K} w_k h_k(\mathbf{x}). \tag{14}$$

$f_K(\mathbf{x})$ is essentially a linear combination of the so-called base models $h_k(\mathbf{x})$'s.

When a (two-class) classification is considered, the combined model becomes a combined classifier $C(\mathbf{x})$, where

$$C(\mathbf{x}) = I(f_K(\mathbf{x})). \tag{15}$$

$I(z)$ is an indicator function, where $I(z) = 1$ for $z > 0$, and $I(z) = 0$ otherwise. $C(\mathbf{x})$ is a label assigned to a feature vector $\mathbf{x}$ by a combined classifier. When $h_k(\mathbf{x})$'s are also classifiers, $C(\mathbf{x})$ is a combination of classifiers.

There are two key issues in combinations of models.

1) How to obtain a set of base models, $\{h_k(\mathbf{x})\}_{k=1}^K$.
2) Given a set of base models, how to choose an optimal set of weighting factors $\{w_k\}_{k=1}^K$ so that the generalization error of the combined model is minimized.

B. Obtaining Base Models

1) Requirements: Numerous algorithms have been developed to explore the first question. These algorithms are distinct in two ways. One results from different architectures for base models and different algorithms for obtaining base models. Another corresponds to different methods used to "perturb" the training process so that the base models obtained can have diverse error patterns. Having classifiers with diverse error patterns were shown to be crucial to effectively improving the performance through combinations of models [61], [63]. This can be understood through two (extreme) examples. At one extreme, if these classifiers are identical, there will be no gain from any combination. At the other extreme, if these classifiers make independent errors with a probability less than 0.5, the overall number of errors made after a combination can decrease exponentially as the number of such classifiers increases [61]. However, the "diversity" is not easy to achieve because all models are trained to do essentially similar tasks. Therefore, a certain randomness should be introduced to "perturb" the learning procedure so that models can learn different parts of a problem, and thereby make errors as differently as possible.

2) Diversification of Base Models: The common methods used to perturb training were categorized by Dietterich [33], [34] into the following cases.

1) Manipulating the input features, i.e., selecting different features for training different base models (classifiers). One such example is Adaboot [45], where every individual classifier is trained on only one randomly chosen dimension of a feature vector.
2) Randomizing training procedures. For example, [84] sets randomized initial weights of neural networks; [57] generates hyperplanes randomly.
3) Manipulating the labels in a training set, i.e., adding the random noise to the labels [32].
4) Manipulating a training set, i.e., resampling or reweighting the original training set from a certain resampling probabilities (or reweighting schemes) [20], [30], [44].

A successful algorithm for generating base models may use one or several aforementioned techniques to diversify base models.

3) Algorithms: Among the above techniques, the most appealing is to randomize training data by selecting a resampling probability due to its effectiveness and simplicity.

Three algorithms to control resampling probabilities have been proposed and proven to be effective by extensive experiments. The first algorithm was originally proposed by Wolpert [102] and was extensively investigated by Breiman [20], [21] as the so-called bagging algorithm. This algorithm generates a training set for training a base model through resampling the original training set with replacement uniformly. Through this resampling algorithm, a newly generated training set is not exactly the same as the original one, although both data sets are drawn from the same distribution. Therefore, the resulting base models from different training sets are different.

The second algorithm was proposed by Freund *et al.* [45]. The algorithm starts from a uniform resampling probability denoted as $p^0(n)$ for a training sample n. The kth model is trained on the kth training set, which is resampled with replacement from the original training set according to probability $p^k(n)$. $p^k(n)$ changes dynamically as follows. If the kth sample is incorrectly classified by the current combined model, $p^k(n)$ is increased from the previous resampling probability $p^{k-1}(n)$. Otherwise it remains the same. Comparing with the above uniformly resampling algorithm, the resampling probabilities here are dynamically changed based on the performance of all previous obtained models.

The third algorithm was proposed by Ji and Ma [30], [57]. This algorithm divides training samples into two classes: a set of "cares" and a set of "don't cares." The former consists of training samples that have not been classified correctly. The latter contains the training samples that have been classified correctly. A base model is then obtained using a set of "cares" alone. This algorithm can be viewed as a compromise between the first two algorithms, since it dynamically determines the set of "cares" and "don't cares" but uniformly resamples only on all "cares."[7]

C. Combination

The other important issue for combination is how to determine the weighting factors. Intuitively, the outputs of base models $\{h_k(\mathbf{x})\}_{k=1}^K$ can be used to "train" the weighting factors $\{w_k\}_{k=1}^K$. However, the straightforward minimization approach often results in overfitting [75], [77] and is therefore not applicable. This is because the base classifiers are highly correlated since they are designed to solve similar tasks. One simplest algorithm called majority vote uses the equal weighting for base models, i.e., $w_k = 1/K$ for all k. Clearly, this scheme is not optimal because it does not take full consideration of differences among base models. But majority vote has been used widely due to its simplicity [20], [22], [57]. More elaborate algorithms have been investigated for combination. Wolpert [102] proposed combining base models through minimizing the squared error using cross validation. Breiman [21] later imposed proper constraints on the weights w_k's. Freund *et al.* [44], [45] proposed using the confidence of a base model as the weight for that model. Principal component analysis [76] was also investigated to explore and thus discount correlation among individual base models.

D. Open Questions

Both theoretical and empirical results [20], [22] suggest that combinations of well-trained models can ease the bias-variance dilemma and improve the performance substantially. That is, we can select a base model with a relatively large size so that it has a small bias but a large variance. A combination scheme can be responsible for reducing the overall variance for a combined model [20], [22], [76], [84]. Therefore, finding an optimal structure of a model is no longer important. Very often, however, the price paid for the gain in performance is a larger space complexity. In addition, as several models are combined, each of which may take a long time to train, an even longer training time results in for a combination. Since training time is critical for real-time applications and is more difficult to tackle than the space complexity, a natural question to ask is whether combinations of models can be used to improve the time complexity at a reasonable cost of the space complexity. Combining weak models provides a promising answer to this question.

[7] Several popular incremental learning algorithms [23], [36], [46] can also be considered as special cases in this class of algorithms.

VI. Performance and Efficiency: Combinations of Weak Models (Classifiers)

Although individual base models in a combination can be quite general, we focus on classifiers as base models in this section.

A. Three Approaches

There are three main approaches developed on combinations of weak classifiers: the boosting algorithm called Adaboost (adaptive boosting algorithm) by Freund and Shapire [44], [45]; the stochastic discrimination (SD) by Kleinberg [61], [62]; and the combination of weak perceptrons (CWP) by Ji and Ma [30], [56], [57].

The concept of weak learning was first introduced by Kearns and Valiant [60] as a part of a theoretical question in the context of the probabilistic approximately correct (PAC) learning theory [13], [97]. The question can be informally stated as "Does the existence of a weak learner imply the existence of an efficient strong learner?" Schapire [93] first proved the existence with a "yes" through a recursive and constructive approach. He showed that if a classification problem is solvable in the PAC framework, the problem can be solved through combinations of weak classifiers that can do a little better than random guessing. Later, Freund [44] showed that in theory a combination of weak classifiers through the simple majority vote can combine the weak classifiers into a strong classifier. The Adaboost [44] was further proposed as a practical algorithm for combinations. Although this work [44], [45] illustrated that weak classifiers could be combined to achieve what a strong classifier could do, it did not further address whether combinations of weak classifiers had any advantage in performance or efficiency as compared with training an individual strong classifier or combining well-trained individual strong classifiers.

In an independent work, Kleinberg [61] proposed stochastic discrimination and showed that combining a large number of weak classifiers could improve the (training) performance monotonically. In addition, he also showed that the time-complexity of combined weak classifiers was polynomial. The derived theory, however, was built on an assumption that weak classifiers made independent classification errors. Such an assumption cannot be achieved in a real situation.

Ji and Ma [30], [56], [57] proposed combinations of weak perceptrons (or neurons). Through extensive experiments, this work showed that a very simple algorithm for generating and combining weak classifiers may achieve better efficiency and performance than training a strong classifier or combining well-trained classifiers. Such a simple procedure is to generate randomly weak perceptrons from the perceptron space as weak classifiers and combine these weak perceptrons through a majority vote. Time-complexity and space-complexity of a combined classifier were formally defined and shown to be polynomial for a "special case" when the strength of weak classifiers was properly chosen. A tradeoff was explicitly shown to exist between time complexity and space complexity.

In the following, we further review these three algorithms and the corresponding theoretical results.

B. Weak Classifiers

1) Definition: The strength of a classifier $C(\mathbf{x})$ can be characterized by ν, the so-called weakness factor, where $\nu > 2$. Let $(1/2) - (1/\nu)$ be the required (generalization) error of classifier $C(\mathbf{x})$, i.e., $\Pr(C(\mathbf{x}) \neq t) = (1/2) - (1/\nu)$. If $\nu >> 2$, the classifier is considered to be a weak classifier because it only performs a little better than random guessing. The larger the ν is, the weaker the weak classifier. The time compelxity and space complexity of $C(\mathbf{x})$ follow the general definition given in Section II but should take into account the weakness factor.

If the space complexity and time complexity of a combined classifier scales polynomially with respect to both the dimension of feature vectors and parameters such as the weakness factor and a desired generalization error, the classifier is said to be efficient. Otherwise, if an exponential scaling is observed, they are considered to be inefficient.

A set of weak classifiers should satisfy the following two conditions: 1) each weak classifier should do better than random guessing and 2) the set of classifiers should have enough computational power to learn a problem. The first condition ensures that each weak classifier possesses a minimum computational power. The second condition suggests that individual weak classifiers should learn different parts of a problem so that a collection of weak classifiers can learn an entire problem. If all the weak classifiers in a collection were to learn the same part of a problem, their combination would not do better than individual classifiers.

2) The Structure of Weak Classifiers: Both local and non-local classifiers have been proposed. Kleinberg *et al.* [62] proposed using a hypersphere as a weak classifier. Such a weak classifier classifies all the samples that fall into the sphere as one class and those outside as the other class. This classifier is local because samples close to the center of a sphere are in the same class. This choice of weak classifiers, however, may suffer from the "curse of dimensionality" and thereby is difficult to handle high-dimensional problems.

Kleinberg [62] and Freund *et al.* [45] independently proposed to use a half space as a weak classifier. A half space classifier classifies input features based on only one selected dimension (or feature) and ignores other dimensions. Therefore, the decision boundary of a half-space classifier is a hyperplane perpendicular to the selected dimension. Such a classifier is global and may be expected to have a nice (polynomial) scaling property with respect to the dimension of feature vectors, thereby avoiding the curse of dimensionality for some cases. As shown by experimental results [44], the set of half-space weak classifiers is limited for representing an arbitrary decision boundary. As a result, the performance is not as good as that of combinations of stronger classifiers. To increase the representation power of the half-space classifiers, the union of two or more half spaces was also considered.

To generate a global classifier with a stronger representational power, Ji and Ma proposed using a perceptron (or a neuron) as a weak classifier [30], [57]. Similar to a half-space classifier, the decision boundary of this classifier is also a hyperplane, and therefore global. However, the orientation of its decision hyperplane can be arbitrary. Therefore, a hyperplane classifier is much more flexible than a half-space classifier. In fact, the combination of these perceptrons (or neurons) forms a two-layer neural network which has been shown to be able to approximate arbitrary functions in theory [10].

C. Algorithms for Combinations of Weak Classifiers

Once the structure of weak classifiers is determined, qualified weak classifiers can be generated and combined.

1) Generation of a Qualified Weak Classifier: There are two fundamentally different approaches to generate a qualified weak classifier. One is to apply a training algorithm to find the "best" weak classifier. The other is to randomly generate a weak classifier from the set of all weak classifiers until a qualified classifier is obtained.

Freund *et al.* [45] used the first scheme, where a feature is first selected randomly and the best threshold is then obtained through exhaustive search. Both stochastic discrimination (SD) [62] and combinations of weak classifiers [57] generated a weak classifier through the second approach. The algorithm can be described as follows:

1) partition the training data into "cares" and "don't cares";
2) randomly generate a candidate classifier from the classifier space;
3) test classification error rate of the candidate classifier on the "cares";
4) if the error rate on the "cared" samples is less than a threshold $0.5 - 1/\nu$, then keep this classifier and go to step 1) to generate another weak classifier; otherwise go to step 2), where ν is the weakness factor.[8]

In this algorithm, a random classifier is first chosen as a candidate classifier. The strength of this candidate classifier is then tested on a set of "cares," which refer to those training samples incorrectly classified. If the candidate passes the threshold, it is accepted and combined with previous qualified classifier. Otherwise, it is rejected and the procedure is repeated until a qualified candidate is found.

[8]A typical value of ν used in [56] is between 50 and 200. A typical number of combined weak classifiers is from 500 to 5000.

Therefore, this approach of generating a qualified classifier can also be called trial until qualified (TUQ).

To generate a perceptron classifier randomly [57], the direction of a hyperplane is first generated randomly and uniformly. Then a feature vector is picked randomly to place the hyperplane in the feature domain.

Three considerations motivate the choice of a randomized (TUQ) approach over the training classifier approach. First, if a qualified classifier is weak enough so that it can be obtained through only several trials, the TUQ may be computational cheaper than training a classifier. Second, the random sampling to obtain a weak classifier may reduce overfitting, which is the problem inherently associated with any training algorithm. Third, random sampling may facilitate theoretical analysis as we will discuss later.

2) Combinations of Qualified Weak Classifiers: In general, the combination schemes developed to combine well-trained models can be used to combine weak classifiers. Specifically, Freund [45] used the weighted majority vote and chose the confidence of the corresponding classifier as the weighting factors. Both combinations of weak perceptrons [57] and stochastic discrimination [62] used the simple majority vote to combine weak classifiers.

3) Adaptively Reweighting and Combining (ARC): Several algorithms for combining either weak or strong models have been discussed. They are bagging [20], Adaboost [44], stochastic discrimination [61], [62], and a combination of weak hyperplanes [30], [57]. Although these algorithms were rooted from different theories and served different purposes, they share common characteristics. Recently, Breiman [22] proposed an ARC framework to capture the common characteristics in these algorithms, where ARC characterizes a combination algorithm as an iterative three-step procedure [assuming $k-1$ $(k \geq 1)$ classifiers have been obtained in the current combination].

1) Resampling (or reweighting) the original training set to obtain the kth training data set. Bagging algorithm uniformly resamples the original training set. Boosting adaptively resamples the original training set. Stochastic discrimination and combinations of weak perceptrons resample uniformly on dynamically chosen "cares."
2) Generating a qualified kth classifier based on the kth training data. Bagging algorithm was designed to combine well-trained models. In practice, both neural networks and decision trees (CART) [20] have been used as base classifiers. Although Adaboost was originally designed to combine weak classifiers, it has been widely used effectively to boost the performance of strong classifiers [22], [87]. SD and CWP all used the TUQ algorithm to generate randomly either a hypersphere or a perceptron but only keep those which exceed a certain performance threshold.
3) Determining the weight for voting. Adaboost uses the confidence over the performance of a model as the corresponding weight for voting. All other algorithms use the simple majority vote combination scheme.

D. Performance of Combinations of Weak Classifiers

Since the performance of Adaboost [45] when used to combine a large number of weak classifiers was found to be either comparable or somewhat inferior to combining strong classifiers, Adaboost algorithms have been used mostly to combine a small number of well-trained classifiers [22], [45], [87]. Through extensive experimental comparisons [22], [45], [87], Adaboost for combining strong models has been shown to perform consistently better than training an individual model or a bagging combination algorithm.

Can combinations of weak classifiers do better in performance than training a strong classifier? Although the independent assumption used in the theory for stochastic discrimination is difficult to satisfy in reality, better performance has been obtained consistently when a large number of weak classifiers are combined and used in handwritten digit recognition [62]. In particular, the resulting combined classifiers were shown to be less sensitive to overfitting. This means the performance of a combined classifier was usually improved when more and more weak classifiers were combined. Similar to combinations of well-trained classifiers, this in fact shows that combinations of weak classifiers reduce the variance when more and more weak classifiers are combined.

Combinations of weak perceptrons [57] were tested extensively on various synthetic and real data sets. The generalization performance of the combined weak perceptrons has been shown to be slightly better than combinations of well-trained classifiers and outperforms individual neural network classifiers and k-nearest neighbor classifiers. Meanwhile, as will soon be shown, interesting phenomena have been observed that a tradeoff can be made between performance and efficiency by combinations of weak perceptrons.

E. Efficiency of Combinations of Weak Classifiers

As the performance of combinations of weak classifiers is comparable to that of combinations of well-trained classifiers, what really are the advantages of using weak classifiers?

1) SD: SD [61] first suggested that combinations of weak classifiers can be used to improve the training time. A theory was derived to show that when weak classifiers were assumed to make independent classification errors, the computational time for selecting a weak classifier was polynomial in terms of the dimension of feature vectors. As the space complexity was also shown to be polynomial, the resulting time complexity of a combined classifier was polynomial. Empirically, the training time needed to obtain a combination of weak classifiers was shown to be magnitudes faster than that of conventional training methods such as BP.

Three factors were not included when the time-complexity was derived: the structure of weak classifiers; the weakness factor; and the statistical dependence among outputs of weak classifiers. As discussed in Section V-A, the structure of weak classifiers relates directly to the space complexity of a combined classifier and thus determines its space efficiency. The weakness factor also affects the space complexity of a combined classifier, since the weaker the weak classifiers are, the more weak classifiers may be needed to learn a problem, and the larger the space complexity. This may in turn influence the time complexity, since the time complexity of a combined classifier can be regarded as the average training time needed to obtain a weak classifier multiplied by the space complexity.[9]

2) *Combinations of Weak Perceptrons:*

a) Empirical results: The efficiency of combinations of weak perceptrons was first tested empirically through various pattern classification problems [30], [57]. For the space complexity, when the k-nearest neighbor classifiers suffer from the curse of dimensionality, a nice scaling property in terms of the number of dimensions was observed for combined weak classifiers.

As for the training time, a qualified weak perceptron could usually be obtained within ten trials for all these experiments. Therefore, combinations of weak perceptrons are magnitudes faster than training a two-layered neural network by BP algorithms.

The empirical results also showed that the weaker the weak perceptrons, the more weak perceptrons (the larger the space complexity of a combined classifier) were needed to achieve a certain performance, and the less time was needed to obtain a weak perceptron. Therefore, a tradeoff may be made between the time and space complexity through a properly chosen weakness factor.

b) Theoretical results: To shed light on whether and why combinations of weak perceptrons are efficient, theoretical analysis was carried out on a simple example when combinations of weak classifiers are used to learn underlying perceptrons [57].

The generalization error $\Pr(C_{2L+1}(\mathbf{x})t < 0)$ of the combined classifier $C_{2L+1}(\mathbf{x})$ with $2L{+}1$ weak perceptrons was derived and bounded above by a quantity in the order of

$$\frac{\nu \ln(2L+1)}{\sqrt{2L+1}}$$

i.e.,

$$\Pr\left(C_{2L+1}(\mathbf{x})t < 0\right) \leq O\left(\frac{\nu \ln(2L+1)}{\sqrt{2L+1}}\right) \tag{16}$$

where ν is the weakness factor. $O(z)$ stands for a quantity in the order of z.

Such a bound shows that the generalization error decreases at a polynomial rate in terms of the number of weak perceptrons. By setting the upper bound to be equal to ϵ_g, which is a bound on the desired generalization error, the space complexity S of a combined classifier[10] can be obtained easily as

$$S \leq O\left(\frac{(\nu \ln \nu)^2}{\epsilon_g^2}\right). \tag{17}$$

[9]The number of weak classifiers by definition.

[10]S is the number of weak classifiers needed to achieve a certain generalization error.

Therefore, S is polynomial in both the weakness of factor ν and ϵ_g.

The time complexity T of a combined classifier is defined as the average number of samplings needed to obtain a combined classifier when a desired generalization error at most ϵ_g. Such a time complexity was shown to satisfy

$$T = O\left(\frac{(\nu \ln \nu)^2}{\epsilon_g^2}\sqrt{d}e^{d/\nu^2}\right) \tag{18}$$

for d and ν large but $\nu \ll d$.

Since the larger the weakness factor ν, the larger the space complexity S, but the smaller the time complexity T, a tradeoff can be made between the space complexity and time complexity by finding an optimal ν. Specifically, let $dT/d\nu = 0$ and assume d large, an optimal weakness factor ν_o can be obtained as $\nu_o = O(\sqrt{d})$. Therefore, when $\sqrt{d/(\ln d)} \leq O(\nu)$, the time complexity T is polynomial in the dimension d of feature vectors; otherwise, T is an exponential function of d. In the meantime, when this condition is satisfied, the space complexity $S = O((d \ln d)/\epsilon_g^2)$ is also polynomial in d. The existence of such a critical value for the weakness factor suggests that the polynomial time complexity may be obtained at a cost of a larger size classifier compared to that of a well-trained classifier with a fixed structure. The cost, however, is theoretically tolerable, since it scales polynomially in the dimension d of feature vectors.

c) Discussion: Through analyzing the time complexity, an intuitive explanation can be drawn on when and why combinations of randomly selected weak perceptrons are efficient. If weak classifiers are weak enough, i.e., the weakness factor ν is large enough, there will be many such weak classifiers. Therefore, the chance of getting a weak classifier is high at each sampling. That is, the number of times needed to sample the classifier space until a qualified weak classifier is accepted is small. As soon as weak classifiers are not too weak to destroy the polynomial space complexity, the efficiency can be achieved both in time and space for combinations of weak perceptrons.

The theory provides a unique explicit relationship between performance and efficiency but is limited to a special case for learning a linear decision boundary. There are no similar results derived so far on nonlinear classification problems.

F. Open Questions

Due to the intrinsic difficulties of tackling performance and efficiency of nonlinear classifiers, many open questions need to be investigated on combinations of weak classifiers. Some of these open questions are given below.

1) How to show the optimality of a (randomized) algorithm for choosing and combining weak classifiers.

   Assuming there were an infinite number of weak classifiers usable in a combination, this question essentially asks whether an algorithm is optimal in approximating a Bayes classifier. As an elegant analysis showed that the nearest-neighbor classifiers [27] are asymptotically Bayesian optimal, it remains open as to whether or not similar results could be obtained for a randomized algorithm using weak classifiers.

2) For what problems do a large number of weak classifiers exist to achieve a desired performance?

   Even when an optimal algorithm is used to obtain a combination of weak classifiers, it is not clear whether a large enough number of weak classifiers exist so that the combined classifier can achieve a desired performance for a given structure (perceptron, for example) and a weakness factor. If the classification problem is too difficult, there may not exist any weak classifier, since a set of weak classifiers with the chosen structure may not have enough capacity [13], [27], [52], [74] to do better than random guessing.

3) For what problems do a large number of weak classifiers exist so that the time complexity can be polynomial?

   Not all problems can be solved in polynomial time, (e.g., training nonlinear neural classifiers is NP complete). A polynomial complexity for randomly selecting a weak classifier is obtained based on the assumption that there exist a large number of weak classifiers. For example, if there exists at least a polynomial fraction of classifiers that can do better than random guessing, the average number of tries required to get one weak classifier is polynomial. Otherwise, choosing one acceptable weak classifier from an exponentially small fraction of all classifiers would take an exponential number of tries on the average. Since the number of acceptable weak classifiers depends on the problem, it is important to characterize problems for which a large number of weak classifiers do exist.

VII. Performance and Efficiency Through Stochastic Training: Evolutionary Algorithms

As explained in the previous sections, randomization is the key for the success of the two combination approaches. In combining well-trained models, randomness is introduced into different phases of the training process to obtain diverse base models. In combining weak classifiers, the randomized algorithm is crucial to generate different weak classifiers to achieve good performance and efficiency. In fact, randomization not only benefits the combination schemes but is also essential to exploring the global rather than local optimization (as deterministic gradient-based approaches do).

In this section, we further elaborate more general randomized algorithms based on evolutionary computation. Evolutionary computation has been used for solving complicated problems in diverse areas of engineering and biology. Its general introduction and related applications can be found in [5], [40]–[42], and references therein. In what follows, we focus on specific applications of evolutionary algorithms to tackling the issues of training neural networks. Other applications can be found in [41], which include applying evolutionary algorithms to finding appropriate network architecture [73], [105] and adaptively adjusting the learning rate [92]. We first illustrate how to apply evolutionary algorithms to train a neural network. We then discuss the similarities and differences between evolutionary algorithms and the algorithm for combinations of weak perceptrons.

A. Evolutionary Computation

A fundamental difficulty of using a deterministic gradient-descent-based algorithm for nonlinear optimization is that the algorithm can be easily trapped by a local minimum. As the error function of a complicated nonlinear system is itself usually very complex and may have many "tiny" local minima and flat areas, using a gradient-based algorithm to train a neural network is usually extremely slow and yields poor results. To ease this problem, evolutionary algorithms, which are stochastic algorithms, were proposed [41] as an alternative to deterministic approaches.

Evolutionary algorithms are a class of stochastic optimization and adaptation techniques that are inspired by natural evolution. They mainly consist of genetic algorithms [51], evolutionary programming [41], and evolution strategies [94]. A comprehensive review of these three different methods can be found in [7]. Although each evolutionary algorithm is designed with a different methodology, all of them are population-based search procedures. When evolutionary algorithms are used to train neural networks, a typical algorithm can be described as follows.

Let $\mathbf{w}$ be a real-valued m-dimensional weight vector of a neural network with an input–output function $f(\mathbf{x};\mathbf{w}) : R^d \to R$, where $\mathbf{x} \in \mathbf{R}^d$ is the input vector, and $f(\mathbf{x};\mathbf{w})$ is the output. The goal is to find a vector $\mathbf{w}$ so that a certain criterion on $f(\mathbf{x};\mathbf{w})$, denoted as $E_f(\mathbf{w})$, can be minimized.

1) Initial step at $t = 0$, and randomly generate an initial population with P parent vectors $\mathbf{w}_i[t]$ for $i = 1, 2, \ldots, P$.

2) Generate O offspring from each parent, i.e.,

$$w'_{i,j}[t] = v(w_i[t]) \tag{19}$$

   where $j = 1, 2, \ldots, O$. v indicates a random variation operation.

3) Select the $(t+1)$th generation based on all offspring $\mathbf{w}'_{i,j}[t]$'s. The fitness of offspring can be evaluated through $E_f(\mathbf{w})$. Then based on the fitness measure, a selection operator s can select a set of offspring as the new parents for the $(t+1)$th generation. That is, a set of the new parents (weight vectors) $\mathbf{W}[t+1]$ satisfies

$$\mathbf{W}[t+1] = s(\mathbf{W}'[t]) \tag{20}$$

   where $(\mathbf{W}'[t]$ is a set of weight vectors consisting of all $w'_{i,j}[t]$'s. Steps 2) and 3) can be combined into one as $\mathbf{W}[t+1] = s(v(\mathbf{W}[t]))$.

4) Let $t = t + 1$. Go back to Step 2) until stopping criteria are satisfied.

At each iteration t which is called a generation, an evolutionary algorithm first generates O offspring (candidates) from each of P parents through a randomization operator v [Step 2)]. As an example, [40] describes a simple variation operation as $v(\mathbf{w}) = \mathbf{w} + \mathbf{z}$, where $\mathbf{z}$ is a vector and its elements are independent Gaussian random variables with zero mean and variance proportional to $E_f(\mathbf{w})$. The next key step [Step 3)] in an evolutionary algorithm is to select a set of offspring as a new generation (qualified candidates). A simple way to select P offspring as the $(t+1)$th generation is to pick the top P offspring based on their performance $E_f(\mathbf{w})$. $E_f(\mathbf{w})$, for instance, can be the one that minimizes an error on a given training set while minimizing the complexity (the number of nonzero weights in w) of the network [10], [39]. More sophisticated variation and selection operators can be found in [41].

B. Efficiency: Relationships to Combinations of Weak Classifiers

To understand how evolutionary algorithms may contribute to tackling the issue of computational efficiency, it is beneficial to compare the similarities and differences between evolutionary algorithms and combinations of weak classifiers.

1) Comparison with Combinations of Weak Classifiers: There are two loops in the algorithm for CWP given in Section VI-C1. The inner loop is used to obtain a qualified weak classifier, and the outer loop is for growing the structure by combining more and more qualified weak classifiers. The algorithm at the inner loop, which is TUQ, can be viewed as a special case of the evolutionary algorithm described above. The outer loop can also be viewed as a special case of evolutionary architecture, where the training samples are evolving toward those which are misclassified.

When weak classifiers are weak perceptrons, $\mathbf{w}$ corresponds to a weight vector that defines a hyperplane. The fitness measure $E_f(\mathbf{w})$ represents the (classification) error rate. The selection threshold is $1/2 - 1/\nu$, which is the upper bound on the training error to ensure that candidate classifiers are not too weak. Corresponding to Step 1), a hyperplane ($\mathbf{w}$) is chosen initially from a uniform distribution. Its strength is then tested using $E_f(\mathbf{w})$. The selection operator s simply keeps those that have a classification error no larger than $1/2 - 1/\nu$. To generate the offspring (hyperplanes at the next step), the randomization operator v simply generates another set of hyperplanes randomly.

However, different from evolutionary algorithms, combinations of weak classifiers combine hyperplanes (parents) at each stage in order to select new hyperplanes (offspring). The final solution for combinations of weak classifiers is a combination of selected parents and offspring at all stages, whereas an evolutionary algorithm intends to find a (single) optimal solution at the final stage. However, the combination can also be included in evolutionary algorithms by revising either the section operator s or the random variation operator v.

Similar comparisons can be made between evolutionary algorithms and other related randomized algorithms given in Section V. Details will be omitted.

2) Discussion: To understand whether and when evolutionary algorithms could be efficient in time, we recall that combinations of weak classifiers use many randomly chosen (weak) classifiers whereas evolutionary algorithms (the current version) intend to find a (single) globally optimal solution. This leads to the question of whether and when it is possible to find a single solution through random sampling in hopefully a polynomial number of steps on the average.

Consider again the simple example on combinations of weak classifiers given in Section VI-E2. When random sampling was used to select a single weak classifier, weak classifiers needed to be weak enough so that the subset of all weak classifiers occupies a large enough fraction in the space of all classifiers defined by one perceptron. Then the average number of times needed to (uniformly) sample the classifier space until a qualified weak classifier was obtained would be polynomial. Since each classifier thus obtained was weak, many such classifiers would need to be combined so that the resulting classifier could have a low classification error. The number of weak classifiers needed in a combination would be polynomial if the classification error of the combined classifiers decreased at least at a polynomial rate with respect to the number of weak classifiers. This shows that the key ingredients for combining randomized weak classifiers are 1) to have a large enough weak classifier space to sample in order to achieve polynomial time-complexity for each weak classifier, and 2) to have a polynomial number of weak classifiers in a combination (a polynomial space complexity) so that the time complexity for a combined classifier is polynomial.

Imagine evolutionary algorithms were used for this case. Then random sampling is used to obtain a single classifier with a low classification error. Since the number of such strong classifiers is an exponentially small fraction of all classifiers, sampling the classifier space randomly will not result in polynomial time complexity. That is, nonuniform sampling is needed in order to achieve polynomial time complexity. This needs to be done by carefully designing the random operator v and the selection operator s so that smaller and smaller classifier subspaces can be sampled to eventually obtain an optimal classifier with a high probability. Finding the operators v and s that can accomplish such a task is crucial to the resulting time complexity. This is equivalent to other problems of random sampling with different samplers [12]. In other words, as this problem is solvable in polynomial time, and extremely simple, it could be used as a simple example to see how an evolutionary algorithm can be designed to achieve the polynomial time complexity.

C. Open Questions

As evolutionary algorithms have shown potential in finding optimal space complexity in a reasonable computational time, little has been done in investigating performance and efficiency of evolutionary algorithms. This, in fact, poses several interesting open questions for possible future research through both empirical and theoretical studies.

1) How can one obtain experimental evidence on the quality of solutions (networks) obtained by evolutionary algorithms to achieve good generalization?
2) How does one rigorously define the time complexity and space complexity of evolutionary algorithms?
3) How does the time-complexity (once defined) scale with the complexity of a benchmark problem (characterized by the dimension of feature vectors)?
4) How should one analyze the performance and efficiency of evolutionary algorithms at least for simple cases?

Due to the complexity of nonlinear optimization, theoretical studies of randomized algorithms would probably be tractable for only simple problems and simplified algorithms. As evolutionary algorithms were posed in a very general form in order to solve complex problems, for analytical tractability, it may be beneficial to consider very simple operators (s and v) for simple examples. In fact, this approach was taken by Karp *et al.*, who introduced randomized algorithms in a general form but provided theoretical analysis on simple cases [15], [29]. He showed that using very simple random sampling, it was possible to obtain a single solution in polynomial time for some cases, which may help analyzing the time complexity of evolutionary algorithms.

For more complex problems such as finding (nonlinear) neural networks, empirical studies on performance and efficiency of evolutionary algorithms on benchmark problems should be possible. Either empirical or theoretical studies on the performance and efficiency of evolutionary algorithms will provide a better understanding of these emerging approaches.

VIII. Conclusion

We have reviewed several general techniques to improve efficiency and performance. Three different approaches have been discussed to improve the performance through making a tradeoff between the bias and variance: 1) searching for an optimal structure for a single network—this was studied mainly for improving the performance of single model; 2) training several oversized models that have a low bias but a high variance, and then diminish the overall variance through combining these models—combinations of well-trained models improve the performance through this approach; 3) training a large set of weak models that have a large bias but a small variance, and then diminish the overall bias and thus the generalization error by combining these weak models—combinations of weak classifiers improve the performance through this scheme. Furthermore, to find global (rather than local) optimal, a stochastic training algorithm, such as an evolutionary algorithm, is essential.

To improve the efficiency for searching for a single optimal solution, the "divide-and-conquer" principle can be used vertically for decision tree algorithms, horizontally for incremental learning algorithms, or recursively for the EM algorithms.

Combinations of weak classifiers, which use an incremental combination scheme and a randomized algorithm, have shown the potential to achieve time efficiency as well as a good generalization performance at a cost of polynomial space complexity for benchmark problems. Explicit relationships have been provided to illustrate the interrelation and the tradeoff between performance and efficiency through combinations of weak classifiers.

As randomized algorithms play an important role for combinations of (weak) classifiers in tackling performance and efficiency, hybrid methods such as evolutionary computation are discussed as a class of more general randomized algorithms.

Although much progress has been made in performance and efficiency of nonlinear adaptive systems, many problems are still wide open for possible future research.

Acknowledgment

The authors would like to thank L. Breiman and A. Barron for pointing out the issue of the existence of weak classifiers, and for helpful discussions. They would also like to thank T. K. Ho, G. Kleinberg, G. Nagy, and G. Zhao for relevant references and helpful discussion. Special thanks are also due to D. B. Fogel for valuable and detailed comments.

References

[1] Y. S. Abu-Mostafa, "Information theory, complexity and neural networks," *IEEE Commun. Mag.*, vol. 27, pp. 25–28, Nov. 1989.

[2] S. Atari, K. Urate, and H. Naga-oka, "Information geometry of Boltzmann machines," *Neural Networks*, vol. 3, pp. 260–271, 1992.

[3] J. A. Benediktsson and P. H. Swain, "Consensus theoretic classification methods," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, pp. 688–704, July–Aug. 1992.

[4] L. Atlas, R. Cole, L. Muthusamy, A. Lippman, J. Connor, D. Park, M. El-Sharkawai, R. J. Marks, "A performance comparison of trained multilayer perceptrons and trained classification trees," *Proc. IEEE*, vol. 78, pp. 1614–1619, Oct. 1990.

[5] W. Atmar, "Notes on the simulation of evolution," *IEEE Trans. Neural Networks*, vol. 5, pp. 130–146, Jan. 1994.

[6] T. Back, *Evolutionary Algorithms in Theory and Practice.* New York: Oxford, 1996.

[7] T. Back, U. Hammel, and H. P. Schwefel, "Evolutionary computation: Comments on the history and current state," *IEEE Trans. Evolutionary Computation*, vol. 1, pp. 1–23, Apr. 1997.

[8] P. Baldi and Y. Chauvin, "Temporal evaluation of generalization during learning in linear networks," *Neural Computation*, vol. 3, pp. 589–603, 1991.

[9] A. Barron, "Approximation and estimation bounds for artificial neural networks," *Machine Learning*, vol. 14, pp. 113–143, 1994.

[10] ——, "Universal approximation bounds for artificial neural networks," *IEEE Trans. Inform. Theory*, vol. 39, pp. 930–944, May 1993.

[11] ——, "Predicted squared error: A criterion for automatic model selection," in *Self-Organizing Methods in Modeling*, S. J. Farlow, Ed. New York: Marcel Dekker, 1984.

[12] ——, personal communication, 1997.

[13] E. Baum and D. Haussler, "What size net gives valid generalization?" *Neural Computation,* vol. 1, no. 1, pp. 151–160, 1989.

[14] E. Baum,"Neural net algorithm that learn in polynomial time from examples and queries?" *IEEE Trans. Neural Networks,* vol. 2, pp. 5–19, Jan. 1991.

[15] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson, "On the power of randomization in on-line algorithms," *Algorithmica*, vol. 1, pp. 2–14, Jan. 1994.

[16] Y. Bengio, "Credit assignment through time: alternative to backpropagation," *NIPS,* vol. 6, pp. 75–82, May 1994.

[17] D. P. Bertsekas and J. Tsitsiklis, *Neuro-Dynamic Programming.* Belmont, MA: Athena Scientific, 1996.

[18] A. L. Blum and R. L. Rivest, "Training a 3-node neural network is NP-complete," *Neural Networks*, vol. 5, pp. 117–127, 1992.

[19] L. Breiman, J. Friedman, and C. Stone, *Classification and Regression Trees.* Belmont, CA: Wadsworth, 1984.

[20] L. Breiman, "Bagging predictors," *Machine Learning,* vol. 24, no. 2, pp. 132–140, Aug. 1996.

[21] ——, "Stacked regressions," *Machine Learning,* vol. 24, no. 1, pp. 49–64, July 1996.

[22] ——, "Arcing classifiers," *Ann. Statistics,* vol. 26, no. 3, pp. 801–849, June 1998.

[23] L. E. Breiman and J. H. Friedman, "Function approximation using ramps," in *Proc. Neural Networks for Computing,* Snowbird, UT, 1993.

[24] T. X Brown, H. Tong, and S. Singh, "Optimizing admission control while ensuring quality of service in multimedia networks via reinforcement learning," to be published.

[25] W. Byne, "Alternating minimization and boltzmann machine learning," *IEEE Trans. Neural Networks,* vol. 3, pp. 612–620, July 1992.

[26] N. Cesa-Bianchi, Y. Freund, D. P. Helmbold, D. Haussler, R. E. Shapire, and M. K. Warmuth, "How to use expert advice," in *Proc. Foundation of Computer Science,* 1993, pp. 382–391.

[27] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inform. Theory,* vol. IT-13, pp. 21–27, 1967.

[28] T. M. Cover, "Capacity problems for linear machines," in *Pattern Recognition,* L. Kanal, Ed. New York: Thompson, 1968.

[29] P. Dagum, R. Karp, M. Luby, and S. Ross, "An optimal algorithm for Monte Carlo estimation," in *Proc. 36th Annu. Symp. Foundations of Computer Science,* Oct. 1995, pp. 142–149.

[30] H. T. Demiral, S. Ma, and C. Ji, "Combined power of weak classifiers," in *Proc. World Congr. Neural Networks,* 1995, pp. 591–595.

[31] A. P. Dempster, N. M. Laird, and D. B Rubin, "Maximum likelihood from incomplete data via EM algorithm," *J. Roy. Statistical Soc. B,* vol. 39, pp. 1–33, 1977.

[32] T. G. Dietterich and G. Bakiri, "Solving multi-class learning problems via error-correcting output codes," *J. AI,* vol. 2, pp. 263–286, 1995.

[33] T. G. Dietterich, "Machine-learning research," *AI Mag.,* vol. 18, no. 4, pp. 97–136, 1997.

[34] ——, "Discussion of Arcing classifiers," *Ann. Statistics,* vol. 26, no. 3, pp. 838–841, June 1998.

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

[36] S. E. Fahlman, "The recurrent cascade-correlation architecture," *NIPS,* vol. 3, pp. 190–196, May 1991.

[37] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From data mining to knowledge discovery: An overview," in *Advances in Knowledge Discovery and Data Mining.* Menlo Park, CA: AAAI/MIT Press, 1996.

[38] T. L. Fine, S. B. Wicker, T. Berger, and J. Halpern, "Sensor-assisted ALOHA for wireless networks," in *Proc. 1998 IEEE Int. Symp. Information Theory,* Aug. 1999, pp. 161–162.

[39] T. L. Fine, *Feedforward Neural Network Methodology.* Berlin, Germany: Springer-Verlag, 1999.

[40] D. B. Fogel, L. J. Fogel, and V. W. Porto, "Evolving neural netowrks," *Bio. Cybern.,* vol. 63, pp. 487–493, 1990.

[41] ——, *Evolutionary Computation: Toward A New Philosophy of Machine Intelligence.* New York: IEEE Press, 1995.

[42] ——, "Practical advantages of evolutionary computation," in *Proc. SPIE,* vol. 3165, pp. 14–22, 1997.

[43] S. Forest, *Emergent Computation: Self-Organizing, Collective, and Cooperative Phenomena in Natural and Artificial Computing Networks.* Cambridge, MA: MIT Press, 1990.

[44] Y. Freund, "Boosting a weak learning algorithm by majority," *Inform. Computation,* vol. 121, no. 2, pp. 256–285, 1995.

[45] Y. Freund and R. Schapire, "Experiments with a new boosting algorithm," in *Machine Learning: Proc. 13th Int. Conf.,* 1996, pp. 148–156.

[46] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: a statistical view of boosting," Dept. of Statistics, Stanford Univ., Stanford, CA, Tech. Rep., 1998.

[47] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural Computation,* vol. 4, no. 1, pp. 1–58, 1992.

[48] R. Grossman, "Learning, capacity and implementation of neural network models," Ph.D dissertation, Wizmann Inst. Sci., 1992.

[49] D. Haussler, M. Kearns, and R. Shapire, "Bounds on the sample complexity of Bayesian learning using information theory and the VC-dimension," in *Proc. 4th Workshop Computational Learning Theory,* 1991.

[50] J. H. Holland, "A mathematical framework for studying learning classifier systems," *Physica,* vol. 22D, pp. 307–317, 1986.

[51] ——,"Genetic algorithms," *Scientific Amer.,* vol. 267, no. 1, pp. 66–72, 1992.

[52] C. Ji and D. Psaltis, "Capacity of two-layer networks with binary weights," *IEEE Trans. Inform. Theory,* vol. 44, pp. 256–268, Jan. 1998.

[53] ——, "The VC-dimension versus the statistical capacity for two layer network with binary weights," in *Proc. 4th Workshop Computational Learning Theory*, 1991.

[54] C. Ji, "Generalization error and the expected network complexity," in *Proc. Neural Information Processing Systems: Natural and Synthetic,* 1993.

[55] C. Ji and D. Psaltis, "Data-driven network synthesis algorithm: addition and deletion," *Neural Networks,* Aug. 1998.

[56] C. Ji and S. Ma. "Combined weak classifiers." *NIPS,* pp. 494–500, May 1996.

[57] ——, "Combinations of weak classifiers," *IEEE Trans. Neural Networks,* vol. 8, pp. 32–42, Jan. 1997.

[58] M. Jordan and R. A. Jacobs, "Hierarchical mixture of experts and the EM algorithm," *Neural Computation,* vol. 6, no. 2, pp. 181–214, 1994.

[59] S. Judd, *Neural Network Design and The Complexity of Learning.* Cambridge, MA: MIT Press, 1990.

[60] M. Kearns and L. G. Valiant, "Cryptographic limitations on learning boolean formulae and finite automata," in *Proc. 21st Annu. ACM Symp. Theory of Computing,* 1989, pp. 433–444.

[61] E. M. Kleinberg, "Stochastic discrimination," *Ann. Math. Artificial Intell.,* vol. 1, pp. 207–239, 1990.

[62] E. M. Kleinberg and T. Ho, "Pattern recognition by stochastic modeling," in *Proc. 3rd Int. Workshop Frontiers in Handwriting Recognition,* Buffalo, NY, May 1993, pp. 175–183.

[63] A. Krogh and J. Vedelsby, "Neural network ensembles, cross validation, and active learning," *NIPS,* pp. 231–238, May 1995.

[64] A. Krogh, G. I. Thorberaso, and J. A. Hertz, "A cost function for internal representations." *NIPS,* vol. 2, pp. 733–745, May 1990.

[65] S. R. Kulkarni, G. Lugosi, and S. S. Venkatesh, "Learning pattern classification-a survey," *IEEE Trans. Inform. Theory,* vol. 44, pp. 2178–2206, Oct. 1998.

[66] S. Lawrence and C. L. Giles, "Searching the Web: general and scientific information access," *IEEE Commun. Mag.,* vol. 37, pp. 116–122, Jan. 1999.

[67] E. Levin, N. Tishby, and S. A. Solla, "A statistical approach to learning and generalization in layered neural network," *Proc. IEEE,* vol. 78, pp. 1568–1574, Oct. 1990.

[68] N. Little and M. Warmuth, "The weighted majority algorithm," in *Proc. 3rd Workshop Computational Learning Theory,* 1989.

[69] S. Ma and C. Ji, "Fast training of recurrent networks based on EM algorithm," *IEEE Trans. Neural Networks,* vol. 9, pp. 11–26, Jan. 1998.

[70] ——, "A unified approach on fast training of feedforward and recurrent networks using EM algorithm," *IEEE Trans. Signal Processing,* vol. 46, pp. 2270–2274, Aug. 1998.

[71] S. Ma, C. Ji, and J. Farmer, "An efficient EM-based training algorithm for feedforward neural networks," *Neural Networks,* vol. 10, no. 2, pp. 243–256, 1997.

[72] D. J. Mackay, "A practical Bayesian framework for backpropagation networks," *Neural Computation,* vol. 4, pp. 448–472, 1992.

[73] V. Maniezzo, "Genetic evolution of the topology and weight distribution of neural networks," *IEEE Trans. Neural Networks,*

vol. 5, pp. 39–53, Jan. 1994.
[74] R. J. McEliece, E. C. Posner, E. R. Rodemich, and S. S. Venkatesh, "The capacity of the Hopfield associative memory," *IEEE Trans. Inform. Theory,* vol. IT-33, no. 4, pp. 461–482, July 1987.
[75] R. Meir, "Bias, variance and the combination of least squares estimators," *NIPS,* vol. 7, pp. 295–302, 1995.
[76] C. Merz and M. J. Pazzani, "Handling redundancy in ensembles of learned models sing principal components." in *Proc. National Conf. Artificial Intell.,* 1996.
[77] ——, "Combining neural network regression estimates with regularized linear weights," *NIPS,* vol. 9, 1997.
[78] Z. Michalewicz, *Genetic Algorithms + Data Structure = Evolution Programs*, 3rd ed. Berlin, Germany: Springer, 1996.
[79] U. Mitra and H. V. Poor, "Neural network techniques for adaptive multiuser demodulation," *IEEE J. Select. Areas Commun.*, vol. 12, pp. 1460–1470, Dec. 1994.
[80] J. Moody and C. Darken, "Fast learning in networks of locally-tuned processing units," *Neural Computation,* vol. 1, pp. 281–294, 1989.
[81] J. Moody, "Generalization, weight decay, and architecture selection for nonlinear learning systems," *Neural Inform. Processing Syst.,* 1991.
[82] R. M. Neal, "Bayesian learning via stochastic dynamics," *Neural Inform. Processing Syst.,* 1993.
[83] S. J. Nowlan and G. E. Hinton, "Simplifying neural networks by soft weight sharing," *Neural Computation,* vol. 4, pp. 473–493, 1992.
[84] M. P. Perrone and L. N. Cooper, "When networks disagree: ensemble method for neural networks," in *Artificial Neural Networks for Speech and Vision.* London, U.K.: Chapman & Hall, 1993, ch. 10.
[85] D. Psaltis, R. R. Snap, and S. S. Venkatesh, "On the finite sample performance of nearest neighbor classifiers," *IEEE Trans. Inform. Theory,* vol. IT-40, pp. 820–837, May 1994.
[86] J. R. Quinlan, *C4.5: Programs for Machine Learning.* San Mateo, CA: Morgan Kaufman, 1993.
[87] ——, "Bagging, boosting, and c4.5," in *Proc. 13th National Conf. Artificial Intelligence,* 1996, pp. 725–730.
[88] S. Rangarajan, P. Jalote, and S. K. Tripathi, "Capacity of voting systems," *IEEE Trans. Software Eng.,* vol. 19, pp. 698–705, 1993.
[89] R. Rohwer, "The moving target training algorithm," *NIPS,* vol. 2, pp. 558–565, May 1990.
[90] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning Internal Representations By Error Propagation.* Cambridge, MA: MIT Press, 1986.
[91] D. Saad and E. Marom, "Learning by choice of internal representations: An energy minimization approach," *Complex Syst.,* vol. 4, pp. 107–118, 1990.
[92] R. Salomon, "Evolutionary algorithms and gradient search: Similarities and differences," *IEEE Trans. Evolutionary Computation,* vol. 2, pp. 45–55, July 1998.
[93] R. Schapire, "The strength of weak learnability," *Machine Learning,* vol. 5, no. 2, pp. 197–227, 1990.
[94] H. P. Schwefel, *Evolution and Optimum Seeking.* New York: Wiley, 1995.
[95] M. Thottan and C. Ji, "Proactive anomaly detection using distributed intelligent agents," *IEEE Network Magazine (Special Issue on Network Management—Today and Tomorrow),* Sept. 1998.
[96] N. Tishby, E. Levin, and S. Solla, "Consistent inference of probability in layered networks: prediction and generalization," in *Proc. Int. Joint Conf. Neural Networks,* 1989.
[97] L. G. Valient, "A theory of learnable," in *Commun. ACM,* vol. 27, no. 11, pp. 1134–1142, 1984.
[98] V. Vapnik, *Estimation of Dependences Based on Empirical Data.* New York: Springer-Verlag, 1982.
[99] ——, *The Nature of Statistical Learning Theory.* New York: Springer Verlag, 1995.
[100] A. Weigend and D. E. Rumelhart, "Generalization by weight elimination with application to forecasting," in *Proc. Int. Joint Conf. Neural Networks,* 1991, pp. 2374–2379.
[101] A. S. Weigend and D. E. Rumelhart, "The effective dimension of the space of hidden units," in *Proc. Int. Joint Conf. Neural Networks*, 1991, pp. 2069–2074.
[102] D. Wolpert, "Stacked generalization," *Neural Networks,* vol. 5, no. 2, pp. 241–259, 1992.
[103] P. J. Werbos, "Supervised learning: can it escape local minimum?" in *Proc. Int. Conf. Neural Networks,* 1993.
[104] H. White, "Parametric statistical estimation with artificial neural networks," Univ. California, San Diego, Tech. Rep., Mar. 1992.
[105] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Trans. Neural Networks,* vol. 8, pp. 694–713, May 1997.

Sheng Ma received the B.S. degree in electrical engineering from Tsinghua University, Beijing, China, in 1987. He received the M.S. and Ph.D degrees in electrical engineering from Rensselaer Polytechnic Institute, Troy, NY, in 1995 and 1998, respectively.

Since 1998, he has been with the IBM T. J. Watson Research Center, Hawthorne, NY, as a Research Staff Member. His current research interests are network traffic modeling and control, network/system management, and data mining.

Chuanyi Ji received the B.S. degree (with honors) from Tshinghua University, Beijing, China, in 1983, the M.S. degree from the University of Pennsylvania, Philadelphia, in 1986, and the Ph.D. degree from California Institute of Technology (Caltech), Pasadena, in 1992, all in electrical engineering.

In November 1991, she joined the faculty of Rensselaer Polytechnic Institute, Troy, NY, as an Assistant Professor. She is now an Associate Professor of Electrical Computer and Systems Engineering at the same institution. Currently, she is on sabbatical at Bell Laboratories, Lucent Technologies, Murray Hill, NJ. In the past, she has done reseach in the area of adaptive learning systems, where she has investigated capacity and generalization capability of nonlinear classfiers and neural networks, randomized algorithms for combinations of weak classifiers, and expectation and maximization algorithms for training static and dynamic neural networks. In recent years, her interests have also been in the areas of computer communication networks. She has been involved in modeling and analysis of heterogeneous network traffic using wavelets, network fault and performance management using adaptive systems, and admission control for wireless networks.

Dr. Ji was a reciepient of the Ming-Li Scholarship at Caltech in 1989 and the NSF Early Career Development (CAREER) Award in 1995.