

Evolving Artificial Neural Network Ensembles



© ARTVILLE

Abstract: Using a coordinated group of simple solvers to tackle a complex problem is not an entirely new idea. Its root could be traced back hundreds of years ago when ancient Chinese suggested a team approach to problem solving. For a long time, engineers have used the divide-and-conquer strategy to decompose a complex problem into simpler sub-problems and then solve them by a group of solvers. However, knowing the best way to divide a complex problem into simpler ones relies heavily on the available domain knowledge. It is often a manual process by an experienced engineer. There have been few automatic divide-and-conquer methods reported in the literature. Fortunately, evolutionary computation provides some of the interesting avenues to automatic divide-and-conquer methods [15]. An in-depth study of such methods reveals that there is a deep underlying connection between evolutionary computation and ANN ensembles. Ideas in one area can be usefully transferred into another in producing effective algorithms. For example, using speciation to create and maintain diversity [15] had inspired the development of negative correlation learning for ANN ensembles [33], [34] and an in-depth study of diversity in ensembles [12], [51]. This paper will review some of the recent work in evolutionary approaches to designing ANN ensembles.

Xin Yao

The University of Birmingham, UK

Md. Monirul Islam

*Bangladesh University of Engineering and Technology
BANGLADESH*

1. Introduction

Artificial neural networks (ANNs) and evolutionary algorithms (EAs) are both abstractions of natural processes. Since early 1990s, they have been combined into a general and unified computational model of adaptive systems, i.e., Evolutionary ANNs (EANNs) [52], [54], to utilize the learning power of ANNs and adaptive capabilities of EAs. EANNs refer to a special class of ANNs in which evolution is another fundamental form of adaptation in addition to learning [55]–[59]. The two forms of adaptation, i.e., evolution and learning, in EANNs make their adaptation to a dynamic environment much more effective and efficient. EANNs can be regarded as a general framework for adaptive systems [59], i.e., systems that can change their architectures and learning rules adaptively without human intervention.

Digital Object Identifier 10.1109/MCI.2007.913386

In Memoriam

Dr. Lawrence (Larry) J. Fogel was one of the few leading lights in the world who were able to make fundamental research contributions to evolutionary computation as well as create a successful business based on his inventions. His talent and scientific acumen were obvious from his early days. After obtaining a Bachelor's degree and Master's Degree in Electrical Engineering from New York University and Rutgers in New Brunswick, respectively, he had become the first pioneer of evolutionary programming by the early 1960s, conducting much of the foundational research into the field. His PhD dissertation 'On the Organization of Intellect' truly set him apart as 'the father of evolutionary programming' as it later became the foundation for the first ever book on evolutionary computation entitled 'Artificial Intelligence Through Simulated Evolution' which was co-authored by Alvin Owens and Michael Walsh. The finite state machines that he used as individuals in evolutionary programming were by far the richest and most expressive representation used. His vision of evolving intelligence has resulted in a flourishing research community pursuing his dreams today.

I first met Larry in Melbourne, Australia, in 1993 when he gave an invited keynote talk at the AI'93 Workshop on Evolutionary Computation. His talk inspired me to pursue my research into evolutionary programming and influenced my research in evolutionary computation in many aspects. So many of us in evolutionary computation enjoyed and benefited from talking to him. He was such an insightful colleague and good friend.

Larry's research discoveries also translated into the business world. In 1965, he founded Decision Science, Inc., which was hailed as the first company dedicated to using evolutionary computation techniques in order to achieve 'real-world problem solving'. In 1993, he founded Natural Selection, Inc., which continues to address problems from the fields of industry, medicine and defence using evolutionary computation techniques. Even towards the end of his life, Larry continued to be active, frequently attending international conferences, including the most recent IEEE CEC last year. His life long commitment and achievements have been recognised with numerous awards: he was a Life Fellow of the IEEE, and in 1996 received the Lifetime Achievement Award from the Evolutionary Programming Society. More recently, in 2006 he received the inaugural 2006 IEEE Frank Rosenblatt Technical Field Award for his 'extraordinary and pioneering achievements in computational intelligence and evolutionary computation'. He will be greatly missed by the evolutionary computation community.

EAs have been introduced into ANNs at roughly three different levels: connection weights, architectures, and learning rules. The evolution of connection weights introduces an adaptive and global approach to training, especially when gradient information is unavailable or too costly to obtain for gra-

dient-based training algorithms to work. Architectural evolution enables ANNs to adapt their topologies to different tasks without human intervention [58]. The evolution of learning rules can be regarded as a process of "learning to learn" in ANNs where the adaptation of learning rules is achieved through evolution [59].

There are strong biological and engineering evidences to support the fact that the information processing capability of ANNs is determined by their architectures. Much work has been devoted to finding the optimal or near optimal ANN architecture using various algorithms, including EAs [59]. However, many real world problems are too large and too complex for any single ANN to solve in practice. There are ample examples from both natural and artificial systems that show that an integrated system consisting of several subsystems can reduce the total complexity of the entire system while solving a difficult problem satisfactorily. Many successes in evolutionary computation have demonstrated that already [63]. The success of ANN ensembles in improving a classifier's generalization is a typical example [62].

ANN ensembles adopt the divide-and-conquer strategy. Instead of using a single large network to solve a complex problem, an ANN ensemble combines a set of ANNs that learn to decompose the problem into sub-problems and then solve them efficiently. An ANN ensemble offers several advantages over a monolithic ANN [47]. First, it can perform more complex tasks than any of its components (i.e., individual ANNs in the ensemble). Second, it can make the overall system easier to understand and modify. Finally, it is more robust than a monolithic ANN, and can show graceful performance degradation in situations where only a subset of ANNs in the ensemble performs correctly. There have been many studies in statistics and ANNs that show that ensembles, if designed appropriately, generalize better than any single individuals in the ensemble. A theoretical account of why and when ensembles perform better than single individuals was presented in [12], [51].

Although ensembles perform better than their members in many cases, constructing them is not an easy task. As mentioned in [16], the key to successful ensemble methods is to construct individual members that perform better than random guessing and produce uncorrelated outputs. This means that individual ANNs in the ensemble need to be accurate as well as diverse [23]. Krogh and Vedelsby [27] formally showed that an ideal ensemble is one that consists of highly correct (accurate) predictors that at the same time disagree, i.e., uncorrelate as much as possible. This has also been tested and empirically verified [40], [41].

2. Evolutionary Ensembles

Evolutionary approaches have been used in evolving ANNs for a long time [59]. However, almost all of such approaches follow a common practice where the best individual ANN is selected as the evolved solution. Such a practice basically treats the learning problem as an optimization one, where the fitness

function is maximized (or the error/cost function is minimized). The error/cost function is defined on the training data. It is well known that the ANN with the smallest training error/cost may not be the ANN with the best generalization ability. While little can be done if we use a non-population based approaches since we only have one candidate solution under consideration, we do have a population of candidate solutions if we use evolutionary approaches. It appears that we should make use of the entire population, rather than just the “best” individual, because a population always contains more information than any single individuals [63].

One simple method to make use of the population information in evolving ANNs is to combine all individuals in a population into an integrated system [62]. Different algorithms could be used to produce weights for such combination. To minimize the overhead, linear combination is often used. The experimental results using such combination methods have shown clearly the advantages of the integrated systems, which can outperform the best individual in a population in terms of testing error rate [62]. It is worth noting that such increased performance was obtained entirely by combining the information in the population. No changes were made to the evolutionary process, which was still EPNNet [61]. The rest of this section will review the evolutionary system and combination methods in detail.

2.1 An Evolutionary Design System for ANNs—EPNNet

EPNNet [61] is an automatic ANN design algorithm based on an evolutionary programming (EP) algorithm [18], [20]. It evolves ANN architectures and their connection weights simultaneously. The emphasis was put on evolving ANNs behaviors, instead of their genetic codes. The main structure of EPNNet is shown in Figure 1.

EPNNet relies on novel mutations and a rank-based selection scheme [53]. It does not use recombination in order to minimize the negative impact of the permutation (i.e., competing conventions) problem [6], [9], [22], which not only makes the evolution inefficient, but also makes recombination less likely to produce highly fit offspring. EPNNet uses five mutation operators to evolve ANN weights as well as architectures. These five mutation operators are arranged in a

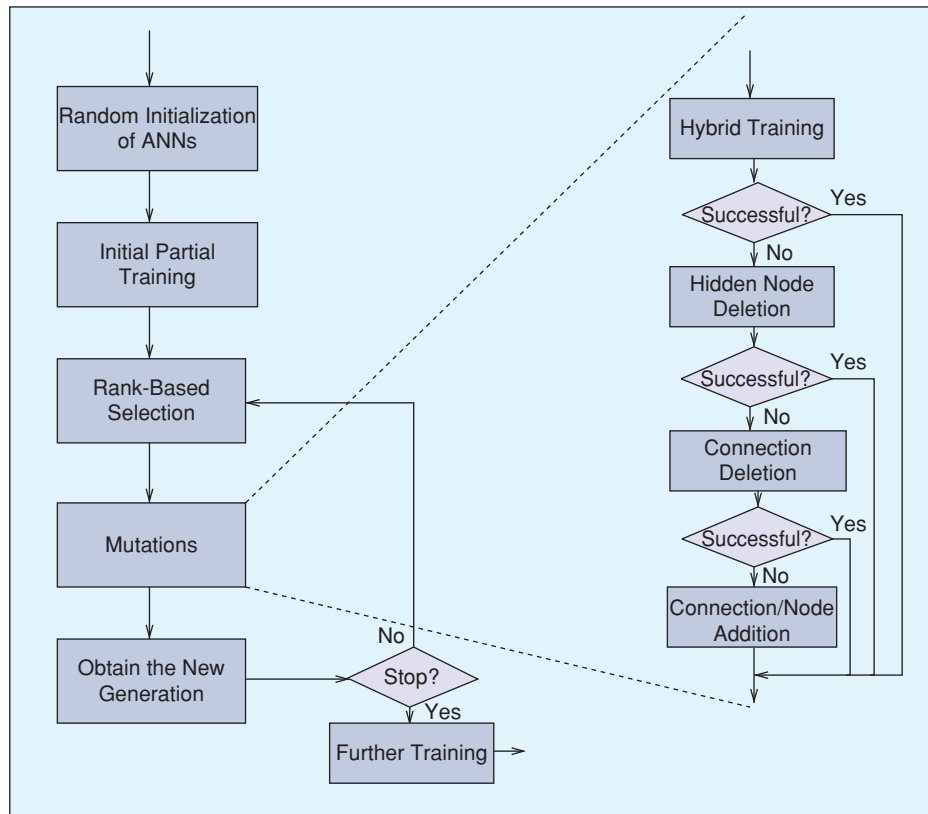


FIGURE 1 Major steps of EPNNet [61].

particular order so that deletion is always attempted before addition. As a result, compact ANNs that generalize can be evolved without adding an additional term in the fitness function to penalize large ANNs. To maintain the behavioral link between parents and offspring, partial training [61] and continuous EP [19] were used. Weight mutation was implemented by a hybrid training scheme consists of a modified back propagation (BP) [46] and simulated annealing. The modified BP can adapt its learning rate for each individual in a population. Simulated annealing is used to avoid the local minimum problem of BP algorithm. The weight mutation is always attempted before any architectural mutations (i.e., node/connection deletion/addition) because the latter cause larger changes in ANN’s behaviors. We want to maintain close links between parents and offspring. Node deletion in EPNNet is done totally at random, i.e., a node is selected uniformly at random for deletion. However, the other three architectural mutations are not uniformly at random. Connection deletion and addition use a nonuniform probability distribution to decide which connection to delete or add based on the importance of the connection [61]. Node addition is achieved by splitting an existing node [39], rather than by introducing a random one. Two nodes obtained by splitting an existing node have the same connections as the existing node. The weights of these new nodes have the following values:

$$\begin{aligned}
w_{ij}^1 &= w_{ij}^2 = w_{ij}, & i \geq j \\
w_{ki}^1 &= (1 + \alpha)w_{ki} & i < k \\
w_{ki}^2 &= -\alpha w_{ki} & i < k
\end{aligned}
\quad
w_i = \frac{\exp(\beta(N + 1 - i))}{\sum_{j=1}^N \exp(\beta_j)} \quad (1)$$

where β is a scaling factor. The ensemble output is

$$O = \sum_{j=1}^N w_j o_j. \quad (2)$$

where \mathbf{w} is the weight vector of the existing node i and \mathbf{w}^1 and \mathbf{w}^2 are the weight vectors of the new nodes, α is a mutation parameter which may take either a fixed or random value, and j and k indicate nodes which have a connection to/from node i .

To improve the generalization ability of evolved ANNs, validation sets are used in EPNet. Each individual (i.e., ANN) is trained on a training set, but its fitness is evaluated on a validation set. All fitness values are calculated based on the validation set, not the training set. After the simulated evolution, all the individuals in the last generation are trained further by the modified BP on the combined training and validation set. A second validation set is used to stop this training and select the best individual as the output of EPNet.

2.2 Combining Methods

As mentioned in the introduction, the best individual in an evolved population can be misleading because the best was defined on the training or validation set. Yet we want the ANN with best generalization ability. Because generalization is hard to measure precisely and accurately in practice, it makes sense not to throw away the second best, third best, etc., in an evolved population. These second or third bests could well be the ANN with the best generalization ability in the future although we cannot know in advance. To exploit as much information as possible in an evolved population without relying on any single individual, we can combine different individuals together.

The four combination methods have been proposed to combine ANNs in a population into an integrated one [62]. The simplest one is majority voting (for classification problems). If there is a tie, the output of the ANN (among those in the tie) with the lowest error rate on the validation set will be selected as the ensemble output. The ensemble in this case is the whole population. All individuals in the last generation participate in voting. The greatest advantage of majority voting is its simplicity. However, the problem of majority voting is that it treats all individuals equally though they are not equally good.

One way to consider differences among individuals without involving much extra computational cost is to use the fitness information to compute a combination weight for each individual. Rank-based linear combination method is such a scheme that attaches a combination weight to each ANN in the population based on their fitness values. In particular, we can use ANN's ranking to generate combination weights. That is, given N sorted ANNs with an increasing error rate, where N is the population size, and their outputs o_1, o_2, \dots, o_N , the weight for the i th ANN is [62]

One of the well-known algorithms for learning linear combination weights (i.e., one-layer linear networks) is the RLS algorithm [38]. The idea behind RLS is to minimize a weighted least squares error. The benefit of using the RLS algorithm is that it is computationally efficient due to its recursive nature.

In the above three combination methods, all the individuals in the last generation of the evolution were used in forming ensembles. It is interesting to investigate whether one can reduce the size of the ensembles without increase testing error rates too much. Such investigation can provide some hints as to whether all the individuals in the last generation contain some useful information.

As the space of possible subsets of a population is very large ($2^N - 1$) for population size N , it is impractical to use exhaustive search to find an optimal subset. Instead, genetic algorithm (GA) [21] can be used [62] to search for a near-optimal subset. The combination weights for each ANN in each subset were determined by the RLS algorithm [38]. The following subsection reviews the actual experimental results by using the previous four combination methods.

2.3 Experimental Studies

Three real-world problems were used in the experimental studies [62]. They were Australian credit card, diabetes and heart disease. The data sets for these problems were obtained from the UCI machine learning repository [8]. There are 690 examples in Australian credit card data set. The output has two classes. The 14 input attributes include six numeric values and eight discrete ones. The diabetes problem is also a two class one. It has 500 examples of class 1 and 268 of class 2. There are eight attributes for each example. The data set is one of the most difficult problems in machine learning due to many missing values. The heart problem is to predict the presence or absence of heart disease given the results of various medical tests carried out on a patient. The heart data set has 13 attributes, which have been extracted from a larger set of 75.

Two validation sets were used in all experiments. One validation set i.e., V-set 1, was used in the fitness evaluation. The other validation set, i.e., V-set 2, was used in further training of ANNs in the final generation. The best individual with the minimum error rate on V-set 2 was chosen as the final result for EPNet (as the best ANN). If there was a tie, the individual with the minimum error rate on the combined training set and V-set 1 was the final result. If a tie still existed, the individual with the minimum error on the combined training set and V-set 1 would be the final result. The final individual was

then tested on an unseen testing set. This is the experiment without using any combination methods and only the best ANN was selected as the output.

2.3.1 Experimental Setup

For all experiments, each data set was randomly partitioned into four subsets, a training set, V-set 1, V-set 2 and a testing set. According to suggestions provided in [43], [44] to produce results for ANNs, the size of the training set, V-set 1, V-set 2, and testing set was chosen to be 50%, 12.5%, 12.5%, and 25% of all examples, respectively in a data set. The input attributes used for all problems were rescaled to between 0.0 and 1.0 by a linear function. The output attributes were encoded using a 1-of- c output representation for c classes. The winner-takes-all method was used to determine the output of ANNs. In this method, the output with the highest activation designates the class.

The same parameters were used for all data set. These were as follows: the population size (20); the maximum number of generations (100); the initial number of hidden nodes (2–8, which means the number of hidden nodes in any initial individual was generated at random between 2 and 8); the initial connection density (0.75, which means the probability of having a connection between two nodes is 0.75; the constraint of feedforward ANNs cannot be violated of course); the initial learning rate (0.2); the number of mutated hidden nodes (1, which means only one node would be deleted/added in each mutation); and the number of mutated connections (1–3, which means the number of mutated connections is between 1 and 3). These parameters were selected after a very modest search. It was found that EPNet [61] was not very sensitive to these parameters.

2.3.2 Results

Table 1 summarizes the results of [62]. The best individual in the last generation and the ensemble formed by four combining methods are presented in the table. The majority voting method outperformed the single best individual on two out of three problems. This is rather surprising since majority voting did not consider the difference among different individuals. It performed worse than the best individual on the heart disease problem probably because it treated all individuals in the population equally. The t -test comparing the best individual to the ensemble formed by majority voting indicates that the ensemble is better than the best individual for the credit card and diabetes problems and worse for the heart disease problem at 0.05 level of significance.

It can be seen from Table 1 that the results of the ensemble formed by rank-based linear method are either better than or

TABLE 1 Testing accuracies of the best individual in a population and ensembles formed from individuals in the population by using majority voting, RLS algorithm [38] and the near optimal subset. The results were averaged over 30 independent runs. Mean, SD, Min and Max indicate the mean value, standard deviation, minimum and maximum value, respectively. Note that the results in the table were summarized from [60].

PROBLEM		ERROR RATE				
		BEST INDIVIDUAL	RANK BASED	MAJORITY VOTING	RLS ALGORITHM	OPTIMAL SUBSET
CARD	MEAN	0.100	0.095	0.095	0.093	0.095
	SD	0.013	0.012	0.012	0.011	0.012
	MIN	0.081	0.070	0.076	0.076	0.070
	MAX	0.128	0.116	0.122	0.116	0.116
DIABETES	MEAN	0.232	0.225	0.222	0.226	0.222
	SD	0.018	0.023	0.022	0.021	0.023
	MIN	0.198	0.172	0.172	0.193	0.182
	MAX	0.271	0.271	0.255	0.260	0.260
HEART	MEAN	0.154	0.154	0.167	0.151	0.164
	SD	0.028	0.031	0.024	0.033	0.030
	MIN	0.103	0.088	0.132	0.088	0.118
	MAX	0.235	0.235	0.235	0.221	0.221

as good as the best individual. The t -test comparing the best individual to the ensemble indicates that the ensemble is better than the best individual for the credit card and diabetes problems at the 0.05 level of significance. The ensemble also outperforms the best individual for the heart disease problem (no statistical significance, however).

The ensemble formed by the RLS algorithm [38] is better than the best individual for all three problems (Table 1). The results also indicate that a better combination method can produce better ensembles. In fact, the RLS algorithm is one of the recommended algorithms for performing linear combinations [24], [42]. However, other algorithms [7] can also be used. The t -test comparing the best individual to the ensemble formed by the RLS algorithm indicates that the ensemble is better than the best individual at the 0.05 level of significance for the credit card and diabetes problems, and better than the best individual at the 0.25 level of significance for the heart disease problem.

The ensemble formed by the subset method is also better than the best individual for the credit card and diabetes problems at the 0.10 and 0.005 level of significance, respectively. It is worse than the best individual for the heart disease problem at the 0.05 level of significance. This worse result might be caused by the small number of generations (only 50) used in the experiments. A large number could probably produce better results, but would increase the computation time.

All the above results indicate that a population contains more information than any single individuals in it. Such information can be used effectively to improve generalization of the learning system. In a sense, the use of population information provides a natural way of evolving modular ANNs, where each module is an individual in the population. It is worth noting that the performance gain of the ensembles over the best individual was obtained without changing anything in the evolution. If we knew we were going to combine all ANNs in

the final generation together, we could improve our evolutionary process by using some additional techniques. After all, our research question has changed from “How to evolve the best individual” into “How to evolve the best population”. It is the entire population, rather than just the best individual, that is evolving.

Since we are taking a more population-focused approach, we could regard individual ANNs in a population as specific modules in a larger integrated system. The evolutionary approach we described above could be regarded as an automatic approach to modularization [15]. One key question here is how to evolve a diverse and accurate set of modules that can be combined together. Speciation in evolutionary computation [14] offers a potential answer. We can use speciated co-evolutionary algorithm to evolve a population of species automatically, where each species could be regarded as a specialist module [15].

3. Automatic Modularisation

Many problems are too large and too complex to be solved by a monolithic system. Divide-and-conquer has often been used to tackle such problems. The key issue here is how to divide a large problem into smaller sub-problems. Tedious trial-and-error processes have often been used by human experts in coming up with a good method for breaking up a large problem into smaller components that are easier to solve. However, it is possible to make use of evolutionary computation techniques to divide a large problem into simpler sub-problems automatically.

Darwen and Yao [15] proposed a novel approach to automatic divide-and-conquer, known as automatic modularisation, in evolving a rule-based system for playing iterated prisoner’s dilemma games without any human intervention. Their results have shown clearly that automatic modularisation can be used to evolve an integrated rule-based system consisting of several sub-systems, each of which is specialised in dealing with certain aspects of a complex problem (e.g., iterated prisoner’s dilemma games). Such sub-systems can be regarded as modules of the integrated system (hence automatic modularisation).

The main idea behind automatic modularisation is a speciated evolutionary algorithm. In the case of evolving game-playing strategies for the iterated prisoner’s dilemma games, each individual in the population is a rule-based system representing a strategy. The implicit fitness sharing scheme used in the speciated evolutionary algorithm will encourage the evolution of species automatically in a population [15]. Each species can be regarded as a sub-system (module) in the integrated system, which is represented by the entire population. The experimental results have shown that automatic modularisation can lead to substantial performance gain in evolving game-playing strategies for the iterated prisoner’s dilemma games [15].

Although the original work on automatic modularisation was done using rule-based systems, the idea is equally applica-

ble to neural networks, decision trees and other classifiers. Khare *et al.* [26] described the most recent work related to automatic modularisation using neural networks.

4. Negative Correlation Learning

Although ANN ensembles perform better than single ANNs in many cases, a number of issues still need to be addressed when using ensembles. Two such important issues are the determination of an ensemble size and the maintenance of diversity among different ANNs in the ensemble. Both theoretical [27], [28] and empirical studies [40], [41] have shown that when individual ANNs are accurate and their errors are decorrelated, improved performance can be obtained by combining the outputs of several ANNs. There is little to be gained by combining ANNs whose errors are positively correlated.

Liu and Yao [33] proposed a new learning paradigm, called negative correlation learning (NCL), for training ANN ensembles. The essence of NCL is that it can produce *negatively* correlated ANNs for an ensemble, which can produce excellent results for many problems [13], [34], [37]). Unlike previous learning approaches for ANN ensembles, NCL attempts to train individual ANNs in an ensemble and combine them in the same learning process. In NCL, all the individual ANNs in the ensemble are trained simultaneously and interactively through the correlation penalty terms in their error functions. Rather than producing unbiased ANNs whose errors are uncorrelated, NCL can create *negatively* correlated ANNs to encourage specialisation and cooperation among the individual ANNs.

Suppose that we have a training set

$$D = \{(\mathbf{x}(1), d(1)), \dots, (\mathbf{x}(N), d(N))\}$$

where $\mathbf{x} \in R^p$, d is a scalar, and N is the size of the training set. The assumption that the output d is a scalar has been made merely to simplify exposition of ideas without loss of generality. Here we consider estimating d by forming an ensemble whose output is a simple averaging of outputs of a set of ANNs:

$$F(n) = \frac{1}{M} \sum_{i=1}^M F_i(n) \quad (3)$$

where M is the number of the individual ANNs in the ensemble, $F_i(n)$ is the output of ANN i on the n th training pattern, and $F(n)$ is the output of the ensemble on the n th training pattern.

NCL introduces a correlation penalty term into the error function of each individual ANN in the ensemble so that all the ANNs can be trained simultaneously and interactively on the same training data set D . The error function E_i for network i in negative correlation learning is defined by

$$E_i = \frac{1}{N} \sum_{n=1}^N E_i(n) \\ = \frac{1}{N} \sum_{n=1}^N \frac{1}{2} (F_i(n) - d(n))^2 + \frac{1}{N} \sum_{n=1}^N \lambda p_i(n) \quad (4)$$

where $E_i(n)$ is the value of the error function of network i at presentation of the n th training pattern. The first term in the right side of Eq. (4) is the empirical risk function of network i . The second term, p_i , is a correlation penalty function. The purpose of minimising p_i is to negatively correlate each ANN's error with errors for the rest of the ensemble. The parameter $0 \leq \lambda \leq 1$ is used to adjust the strength of the penalty. The penalty function p_i may have the form:

$$p_i(n) = (F_i(n) - F(n)) \sum_{j \neq i} (F_j(n) - F(n)) \quad (5)$$

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

$$\frac{\partial E_i(n)}{\partial F_i(n)} = F_i(n) - d(n) + \lambda \frac{\partial p_i(n)}{\partial F_i(n)} \\ = F_i(n) - d(n) + \lambda \sum_{j \neq i} (F_j(n) - F(n)) \\ = F_i(n) - d(n) - \lambda (F_i(n) - F(n)) \\ = (1 - \lambda) (F_i(n) - d(n)) + \lambda (F(n) - d(n)) \quad (6)$$

where we have made use of the assumption that $F(n)$ has a constant value with respect to $F_i(n)$. The standard BP algorithm [46] can be used for weight adjustments in the mode of pattern-by-pattern updating. That is, weight updating of all the individual ANNs is performed simultaneously using Eq. (6) after the presentation of each training pattern. One complete presentation of the entire training set during the learning process is an *epoch*.

NCL from Eq. (6) is a simple extension to the standard BP algorithm. In fact, the only modification that is needed is to calculate an extra term of the form $\lambda (F_i(n) - F(n))$ for the i th network. From Eqs. (4), (5) and (6), we can make the following observations:

1. During the training process, all the individual ANNs interact with each other through their penalty terms in the error functions. Each ANN i minimizes not only the difference between $F_i(n)$ and $d(n)$, but also the difference between $F(n)$ and $d(n)$. That is, NCL considers errors that all other ANNs have learned while training an ANN.
2. For $\lambda = 0.0$, there are no correlation penalty terms in the error functions of the individual ANNs, and the individual ANNs are just trained independently. That is, independent training for the individual ANNs is a special case of NCL.
3. For $\lambda = 1$, from Eq. (6) we get

$$\frac{\partial E_i(n)}{\partial F_i(n)} = F(n) - d(n) \quad (7)$$

Note that the empirical risk function of the ensemble for the n th training pattern is defined by

$$E_{ens}(n) = \frac{1}{2} \left(\frac{1}{M} \sum_{i=1}^M F_i(n) - d(n) \right)^2 \quad (8)$$

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

$$\frac{\partial E_{ens}(n)}{\partial F_i(n)} = \frac{1}{M} \left(\frac{1}{M} \sum_{i=1}^M F_i(n) - d(n) \right) \\ = \frac{1}{M} (F(n) - d(n)) \quad (9)$$

In this case, we get

$$\frac{\partial E_i(n)}{\partial F_i(n)} \propto \frac{\partial E_{ens}(n)}{\partial F_i(n)} \quad (10)$$

The minimization of the empirical risk function of the ensemble is achieved by minimizing the error functions of the individual ANNs. From this point of view, NCL provides a novel way to decompose the learning task of the ensemble into a number of subtasks for different individual ANNs.

4.1 Evolutionary Ensembles with Negative Correlation Learning

Based on NCL [33] and evolutionary learning, Liu *et al.* [34] proposed evolutionary ensembles with NCL (EENCL) to determine automatically the number of individual ANNs in an ensemble and to exploit the interaction between individual ANNs design and their combination. In EENCL, an evolutionary algorithm based on EP [18], [20] was used to search for a population of diverse individual ANNs for constructing ensembles.

Two schemes are used in EENCL to maintain diversity in different individuals (i.e. ANNs) of the population. They are fitness sharing [63] and NCL [33]. The fitness sharing encourages speciation by degrading the raw fitness (i.e., the unshared fitness) of an individual according to the presence of similar individuals. If one training example is learned correctly by n individuals in a population, each of these n individuals receives fitness $1/n$, and the remaining individuals in the population receive fitness zero. Otherwise, all the individuals in the population receive fitness zero. This procedure is repeated for all examples in the training set. The fitness of an individual is then determined by summing its fitness values over all training examples.

EENCL uses Gaussian mutation to produce offspring from parents though non-Gaussian mutation such as Cauchy mutation [64] and Lévy mutation [32] can also be used. The mutation is carried out in two steps: weight mutation and further weight training. In the first step, n_b parent networks are

TABLE 2 Accuracy rates of EENCL for the Australian credit card and diabetes data sets. The results are averaged over 10-fold cross-validation for the Australian credit card data set and 12-fold cross-validation for the diabetes data set. The Mean, SD, Min, and Max indicate the mean value, standard deviation, minimum, and maximum value, respectively [34].

ACCURACY RATE		SIMPLE AVERAGING		MAJORITY VOTING		WINNER-TAKES-ALL	
		TRAINING	TESTING	TRAINING	TESTING	TRAINING	TESTING
CREDIT CARD	MEAN	0.910	0.855	0.917	0.857	0.887	0.865
	SD	0.010	0.039	0.010	0.039	0.007	0.028
	MIN	0.897	0.797	0.900	0.812	0.874	0.812
	MAX	0.924	0.913	0.928	0.913	0.895	0.913
DIABETES	MEAN	0.795	0.766	0.802	0.764	0.783	0.779
	SD	0.007	0.039	0.007	0.042	0.007	0.045
	MIN	0.783	0.703	0.786	0.688	0.774	0.703
	MAX	0.805	0.828	0.810	0.828	0.794	0.844

selected at random to create n_b offspring. The parameter n_b is specified by a user. The following equation is used for weight mutation [34].

$$w'_{ij} = w_{ij} + N(0, 1) \quad (11)$$

where w'_{ij} and w_{ij} denote the weights of offspring i and parent i , respectively, $i = 1, \dots, n_b$, j is the index number of weights. $N(0, 1)$ denotes a Gaussian random variable with mean zero and standard deviation one.

In the second step, the n_b offspring ANNs are further trained by NCL [33]. EENCL selects the fittest M ANNs from the union of M parents ANN and n_b offspring ANN. Here M is the number of individuals in the population. EENCL repeats the process of offspring generation and selection process for the g generation specified by a user.

A population of ANNs is found after the evolutionary process finishes. A question that arises is how to form the ensemble from a population of ANNs. The most convenient way is to use all ANNs, i.e. the whole population in the last generation. The other way is to use a subset of the population by selecting one representative from each species in the last generation. The species in the population can be obtained by clustering the individuals in the population using any clustering algorithm (e.g. *k-means algorithm* [35]). The later way may reduce the size of an ensemble without worsening its perfor-

TABLE 3 Accuracy rates of the ensemble formed by the representatives from species. The results are averaged over 10-fold cross-validation for the Australian credit card data set and 12-fold cross-validation for the diabetes data set. Mean, SD, Min, and Max indicate the mean value, standard deviation, minimum, and maximum value, respectively [34].

ACCURACY RATE	CARD		DIABETES	
	TRAINING	TESTING	TRAINING	TESTING
MEAN	0.887	0.868	0.783	0.777
SD	0.004	0.030	0.009	0.042
MIN	0.881	0.812	0.770	0.719
MAX	0.890	0.913	0.798	0.844

mance too much. In EENCL, both of these two approaches were used and compared.

Three combination methods were used to determine the output of an ensemble from different ANNs used for forming the ensemble. They are simple averaging, majority voting and winner-takes-all. In simple averaging, the output of the ensemble is obtained by averaging the output of individual ANNs in the ensemble. In majority voting, the output of the majority ANNs will be the

output of the ensemble. If there is a tie, the output of the ensemble is rejected. In winner-takes-all, the output of the ensemble is only decided by the individual ANN whose output has the highest activation.

4.1.1 Experimental Studies

EENCL was applied to two benchmark problems: the Australian credit card problem and the diabetes problem. The n -fold cross-validation technique was used to divide the data randomly into n mutually exclusive data groups of equal size. In each train-and-test process, one data group is selected as the testing set, and the other $(n - 1)$ groups become the training set. The estimated error rate is the average error rate from these n groups. In this way, the error rate is estimated efficiently and in an unbiased way. The parameter n was set to be 10 for the Australian credit card data set and 12 for the diabetes data set in order to facilitate the comparison with other algorithms.

The same parameters were used for both problems. They are as follows: the population size 25, the number of generations 200, the reproduction block size n_b 2, the strength parameter λ for NCL 0.75, the number of training epochs 5, the minimum number of cluster sets 3, and the maximum number of cluster sets 25. The ANN architecture is the multilayer perceptron with one hidden layer and five hidden nodes.

4.1.2 Experimental Results

All the results presented in this section is summarized from results presented in [34]. Table 2 shows the results of EENCL for the two data sets, where the ensembles were formed using the whole population in the last generation. The *accuracy rate* refers to the percentage of correct classifications produced by EENCL. When comparing the accuracy rates obtained by three combination methods, winner-takes-all outperformed simple averaging and majority voting on both problems. The main reason may be that winner-takes-all treats different individuals differently, while all individuals are treated equally by simple averaging and majority voting.

The results of the ensemble formed by the representatives from species are given in Table 3. The combina-

TABLE 4 Sizes of the ensembles using the representatives from species. The results are averaged over 10-fold cross-validation for the Australian credit card data set and 12-fold cross-validation for the diabetes data set. *Mean, SD, Min, and Max* indicate the mean value, standard deviation, minimum, and maximum value, respectively [34].

	SIZE OF THE ENSEMBLES			
	MEAN	SD	MIN	MAX
CARD	13.2	7.8	5	25
DIABETES	16.3	6.4	5	25

tion method used is winner-takes-all. The t -test statistics comparing the accuracies of the ensembles using the representatives from species to the ensembles using the whole population are 0.80 for the Australian credit card data set, and -0.36 for the diabetes data set. No statistically significance difference was observed between them for both data sets ($p > 0.05$), which implies that the ensemble does not have to use the whole population to achieve the good performance. The size of the ensemble can be substantially smaller than the population size. The reduction in the size of the ensembles can be seen from Table 4 which gives the sizes of the ensembles using the representatives from species.

5. Constructive Approaches to Ensemble Learning

The determination of ensemble size by an evolutionary approach is presented in section 4.1. The problem with ENNCL [34] is that it only determines the number of individual ANNs in the ensemble automatically, but the architectures of individual ANNs need to be specified by the user. It is well known that the accuracy of ANNs is dependent on their architectures. This means random selection of the architectures of ANNs may hurt the performance of ensembles. This is because the performance of ensembles not only dependent on the diversity of individuals ANNs but also the accuracy of ANNs. The aim of this section is to present a constructive algorithm for training cooperative neural-network ensembles (CNNEs), where both ensemble and individual ANN architectures are determined automatically [37].

Unlike most previous studies on training ensembles, CNNE puts emphasis on both accuracy and diversity among individual ANNs in an ensemble. It uses a constructive approach to determine automatically the number of ANNs in an ensemble and of hidden neurons in ANNs. The automatic determination of hidden neurons ensures accuracy of individual ANNs in designing the ensemble. CNNE trains each individual ANN with a different number of training epoches, which is determined automatically by its training process. Similar to ENNCL [34], it also uses NCL [33] to train individual ANNs so that they can learn different aspects or parts of the training data. The use of NCL and different training epochs reflects CNNEs emphasis on diversity among individual ANNs in the ensemble.

A number of issues, such as the number of individual ANNs in an ensemble, the number of hidden nodes in ANNs and the number of epochs required for training ANNs, need

to be addressed when designing ensembles. This means that the design of ANN ensembles could be formulated as a multi-objective optimization problem. CNNE uses a simple approach based on incremental training in designing ensembles. It tries to minimize the ensemble error first by training a minimal ensemble architecture, then by adding several hidden nodes one by one to existing ANNs and lastly by adding new ANNs one by one. The minimal ensemble architecture consists of two ANNs with one hidden layer in each ANN and one node in the hidden layer of ANNs.

The main structure of CNNE is shown in Figure 2 and the detailed description can be found in [37]. CNNE uses simple criteria for adding hidden nodes and ANNs. The criteria is based on contribution of ANNs to the ensemble. The following equation is used to determine the contribution.

$$C_i = 100 \left(\frac{1}{E} - \frac{1}{E^i} \right), \quad (12)$$

where E is the ensemble error *including* individual ANN i and E^i is the ensemble error *excluding* individual ANN i . CNNE adds hidden nodes to an individual ANN when its contribution to the ensemble does not improve much after a certain amount of training. An individual ANN is added to the ensemble when adding several hidden nodes to the previously added have failed to reduce the ensemble error significantly. When a new ANN is added to the ensemble, CNNE stops the construction process of the previously added ANN. This means that no hidden node will be added to the previously added ANN in future.

5.1 Experimental Studies

CNNE was applied to seven benchmark problems: the Australian credit card problem, the breast cancer problem, the diabetes problem, the glass problem, the heart disease problem, the letter recognition problem, and the soybean problem. The datasets representing these problems were obtained from the UCI machine learning benchmark repository. For all our experiments, each data set was partitioned into three subsets, a training set, a validation set and a testing set. The size of the training set, validation set, and testing set was 50%, 25%, and 25% of all examples, respectively. The only exception is the letter data set, where 16000 and 2000 examples were randomly selected from 20000 examples for the training and validation sets, and the remaining 2000 examples were used for the testing set.

Initial connection weights for individual ANNs in an ensemble were randomly chosen in the range between -0.5 and 0.5 . The learning rate and momentum for training individual ANNs were chosen in the range of 0.10 – 0.50 and 0.5 – 0.9 , respectively. The number of training epochs for partial training [61] of individual ANNs was chosen to be between 5 and 25. The number of hidden nodes used for halting the construction of individual NNs was chosen to be between one and five. The threshold value ϵ was chosen to be

between 0.10 and 0.20. These parameters were chosen after some preliminary experiments. They were not meant to be optimal. The parameter λ used to adjust the strength of the penalty term in NCL was set to be 1.0.

5.1.1 Experimental Results

Table 5 shows the results of CNNE over 30 independent runs on the seven problems. The results presented in the table are

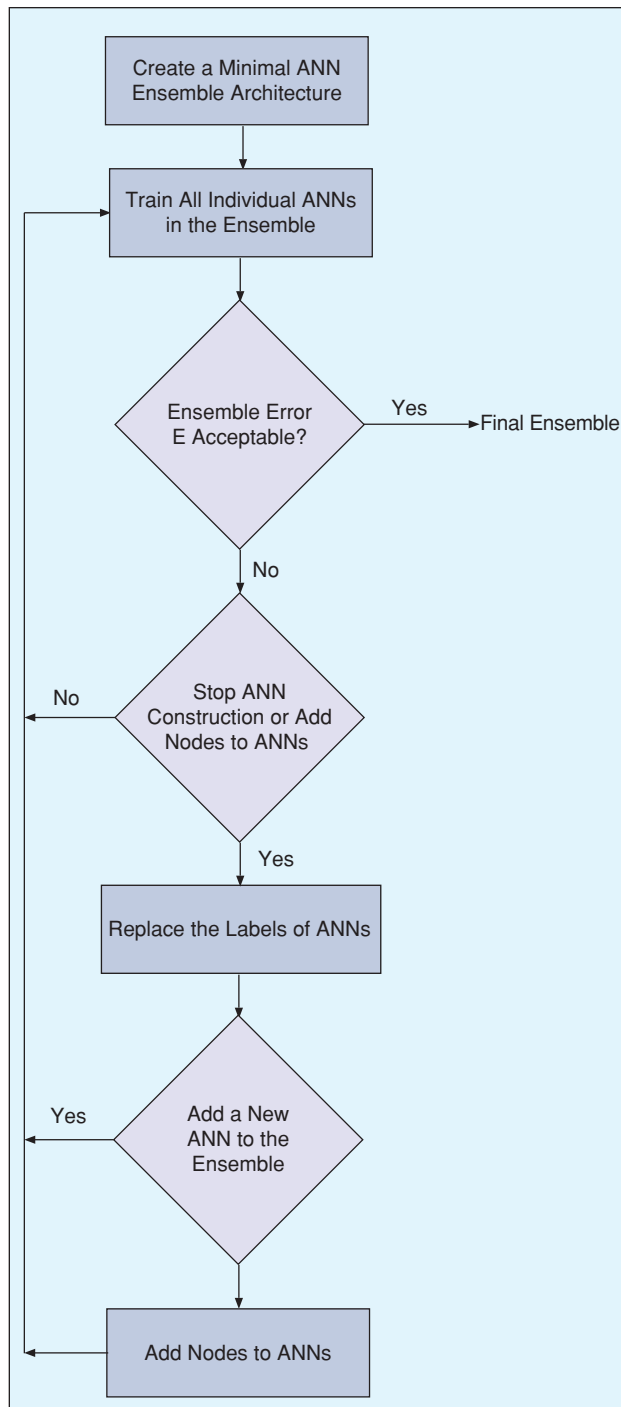


FIGURE 2 Major steps of CNNE [37].

summarized from [37]. The error rate in the table refers to the percentage of wrong classification produced by the trained ensemble on the testing set. M and N in the table represent the number of ANNs in an ensemble and that of hidden nodes in an ANN, respectively.

It can be seen from Table 5 that the ensemble architectures learned by CNNE were influenced by the values of user specified parameters τ and m_h . For example, for the card problem when $\tau = 10$ and $m_h = 4$, the average number of individual ANNs and hidden nodes were 6.5 and 5.3, respectively, while the average number of individual ANNs and hidden nodes were 7.8 and 4.7, respectively, when $\tau = 10$ and $m_h = 2$. These results indicate that, for the same value of τ , the number of individual ANNs in an ensemble increases when the number of hidden nodes in ANNs decreases. This is reasonable because a small ANN has only a limited processing power. CNNE added more ANNs to the ensemble when the size of individual ANNs was small. However, it is worth noting that the testing error rate remained roughly the same for different parameter settings and different ensemble architectures. The choice of different parameters did not affect the performance of the learned ensembles much, which is a highly desirable feature for any ANN training algorithm.

CNNE's ability of constructing different ensembles for different problems automatically can be seen clearly from Table 5. CNNE produced large ensembles for the letter problem, which is large in comparison with other problems we have, and smaller ensembles for other problems. However, there are other factors in addition to the size of training sets, e.g., complexity of the given problem and noise in the training set, that influence the ensemble architecture.

6. Multi-objective Approaches to Ensemble Learning

As mentioned previously, ensemble learning could be formulated as a multi-objective optimization problem. The aim of this section is to introduce multi-objective evolutionary approaches to ensemble learning. The idea of designing ANNs using a multi-objective evolutionary approach was first considered by Abbas [3]. In [3], a new algorithm, called memetic Pareto artificial neural network (MPANN) is proposed for training ANNs. It combines multi-objective evolutionary algorithm and a gradient-based local search in reducing network complexity and training error. MPANN was later applied to learning and formation of ANN ensembles with a different multi-objective formulation [4], [5].

When a population of ANNs is evolved using a multi-objective evolutionary approach, different ANNs in the population may be good for different objectives. This means that we are getting a set of pareto optimal ANNs that can easily be used for constructing ensembles.

Recently Chandra and Yao [13] proposed an algorithm, called diverse and accurate ensemble learning algorithm (DIVACE), that uses multi-objective evolutionary approach to ensemble learning. DIVACE tries to find an optimum trade-off between diversity and accuracy by treating them explicitly

TABLE 5 Architectures and accuracies of ensembles produced by CNNE for the seven classification problems. The results were averaged over 30 independent runs. M and N indicate the number of ANNs in an ensemble and of hidden nodes in an ANN, respectively [37].

	$\tau = 10, m_h = 4$			$\tau = 10, m_h = 2$			$\tau = 15, m_h = 2$		
	M	N	ERROR RATE	M	N	ERROR RATE	M	N	ERROR RATE
CARD	6.5	5.3	0.090	7.8	4.7	0.092	7.4	4.3	0.091
CANCER	3.9	3.6	0.015	4.8	2.9	0.013	4.5	2.5	0.012
DIABETES	4.7	4.5	0.201	6.5	3.4	0.198	6.2	3.2	0.196
GLASS	4.9	4.6	0.261	6.2	3.8	0.268	6.0	3.5	0.258
HEART	4.6	6.5	0.140	5.5	4.9	0.134	5.8	4.2	0.138
LETTER	11.6	10.6	0.067	15.3	8.5	0.062	13.9	8.1	0.060
SOYBEAN	5.3	5.5	0.081	7.1	4.2	0.076	6.8	3.8	0.078

as two separate objectives. The evolutionary process of DIVACE is quite similar to that used in pareto differential evolution [1] and in MPANN [4], [5]. It has three main components, i.e., fitness evaluation, selection and evolutionary operations.

The fitness evaluation and selection in DIVACE are based on non-dominated sorting procedure proposed by Srinivas and Deb [49]. Three parent recombination and mutation are used in the evolutionary algorithm, i.e., a variant of differential evolution [50]. DIVACE also incorporates the idea of simulated annealing. The following equations are used to produce offspring through recombination.

$$w_{hi} = w_{hi}^{\alpha_1} + N(0, \sigma^2)(w_{hi}^{\alpha_2} - w_{hi}^{\alpha_3}) \quad (13)$$

$$w_{oh} = w_{oh}^{\alpha_1} + N(0, \sigma^2)(w_{oh}^{\alpha_2} - w_{oh}^{\alpha_3}) \quad (14)$$

where w_{hi} and w_{oh} are the weights (input to hidden layer and hidden to output layer respectively) of the child generated. α_1 , α_2 and α_3 indicate three parents. Mutation is applied to offspring generated by recombination with probability $1/N$, where N is the size of the population. The following equations are used for mutation.

$$w_{hi} = w_{hi} + N(0, 0.1) \quad (15)$$

$$w_{oh} = w_{oh} + N(0, 0.1) \quad (16)$$

6.1 Experimental Studies

This section presents some results obtained on evaluating DIVACE on the Australian credit card and diabetes problems.

TABLE 6 Performance (accuracy rates) of the ensemble formed using DIVACE on the Australian credit card dataset [13].

	AUSTRALIAN CREDIT CARD ASSESSMENT DATASET					
	SIMPLE AVERAGING		MAJORITY VOTING		WINNER-TAKES-ALL	
	TRAINING	TESTING	TRAINING	TESTING	TRAINING	TESTING
MEAN	0.872	0.862	0.867	0.857	0.855	0.849
SD	0.007	0.049	0.007	0.049	0.007	0.053
MAX	0.884	0.927	0.879	0.927	0.864	0.927
MIN	0.859	0.753	0.856	0.768	0.842	0.753

TABLE 7 Performance (accuracy rates) of the ensemble formed using DIVACE on the Diabetes dataset [13].

	DIABETES DATASET					
	SIMPLE AVERAGING		MAJORITY VOTING		WINNER-TAKES-ALL	
	TRAINING	TESTING	TRAINING	TESTING	TRAINING	TESTING
MEAN	0.780	0.773	0.783	0.766	0.766	0.766
SD	0.006	0.050	0.005	0.057	0.017	0.049
MAX	0.791	0.859	0.791	0.875	0.796	0.843
MIN	0.768	0.687	0.772	0.671	0.730	0.671

The experimental setup is similar to that in [4], [5] in order to facilitate comparison with previous work and for consistency. We used 10-fold and 12-fold cross validation for the credit card and diabetes problems, respectively. Three combining methods, i.e. simple averaging, majority voting and winner-takes-all are used in the experiments.

6.1.1 Experimental Results

Table 6 shows the performance accuracy of the formed ensemble on the Australian credit card assessment dataset. Table 7 shows the same for the Diabetes dataset. Good performance can be observed for the DIVACE algorithm in both cases.

7. Conclusions

Combining ANNs with evolutionary computation has been a popular topic since late 1980s. While the early work tended to focus on evolving single ANNs networks, at the level of weights, architectures and learning rules, recent work has

moved towards evolving ANN ensembles. This is a natural trend because it is often impractical to evolve or design a monolithic ANN when the problem to be solved gets larger and more complex. Divide-and-conquer strategy must be used in practice. ANN ensembles can be regarded as an effective approach to implement the divide-and-conquer strategy in practice. Evolutionary computation provides a powerful method for evolving such ensembles automatically, including automatic determination of weights, individual ANN architectures and the ensemble structure.

References

[1] H.A. Abbass, and R. Sarker, and C. Newton, "PDE: A Pareto-frontier differential evolution approach for multi-objective optimization problems," *Proc. of the IEEE Congress on Evolutionary Computation (CEC2001)*, IEEE Press, pp. 971–978, 2001.

[2] H.A. Abbass, "The self-adaptive pareto differential evolution algorithm," *Proc. of the 2002 Congress on Evolutionary Computation CEC2002*, IEEE Press, pp. 831–836, 2002.

[3] H.A. Abbass, "Speeding up backpropagation using multiobjective evolutionary algorithms," *Neural Computation*, vol. 15, no. 11, pp. 2705–2726, 2003.

[4] H.A. Abbass, "Pareto neuro-evolution: constructing ensemble of neural networks using multi-objective optimization," *Proc. of the 2003 Conference on Evolutionary Computation*, IEEE Press, pp. 2074–2080, 2003.

[5] H.A. Abbass, "Pareto neuro-ensemble," *Proc. of the 16th Australian Joint Conference on Artificial Intelligence*, Springer, pp. 554–566, 2003.

[6] P.J. Angeline, G.M. Saunders, and J.B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Trans. Neural Networks*, vol. 5, no. 1, pp. 54–65, 1994.

[7] P.F. Baldi and K. Hornik, "Learning in linear neural networks: A survey," *IEEE Trans. Neural Networks*, vol. 6, no. 4, pp. 837–858, 1995.

[8] C. Blake and C. Merz, UCI repository of machine learning databases. <http://www.ics.uci.edu/~mllearn/MLRepository.html>.

[9] R.K. Belew, J. McInerney, and N.N. Schraudolph, "Evolving networks: Using genetic algorithm with connectionist learning," *Tech. Rep. CS90-174 (revised)*, Comput. Sci. Eng. Dept. (C-014), Univ. Calif., San Diego, Feb. 1991.

[10] D. Bollé, D.R.C. Dominguez, and S. Amari, "Mutual information of sparsely coded associative memory with self-control and tenary neurons," *Neural Networks*, vol. 13, pp. 452–462, 2000.

[11] G. Brown and J.L. Wyatt, "Negative correlation learning and the ambiguity family of ensemble methods," *Proc. Int. Workshop on Multiple Classifier Systems*, Springer, pp. 266–275, 2003.

[12] G. Brown, J. Wyatt, R. Harris, and X. Yao, "Diversity creation methods: A survey and categorisation," *J. Inf. Fusion*, vol. 6, pp. 5–20, 2005.

[13] A. Chandra and X. Yao, "Ensemble learning using multi-objective evolutionary algorithms," *Journal of Mathematical Modelling and Algorithms*, vol. 5, no. 4, pp. 417–445, 2006.

[14] P. Darwen and X. Yao, "Every niching method has its niche: Fitness sharing and implicit sharing compared," in *Proc. of Parallel Problem Solving from Nature (PPSN IV)*, vol. 1141 of Lecture Notes in Computer Science, Berlin, Germany: Springer-Verlag, pp. 398–407, 1996.

[15] P.J. Darwen and X. Yao, "Speciation as automatic categorical modularization," *IEEE Transactions on Evolutionary Computation*, vol. 1, pp. 101–108, 1997.

[16] T.G. Dietterich, "Machine-learning research: four current directions," *AI Mag.*, vol. 18, no. 4, pp. 97–136, 1998.

[17] W. Finnoff, F. Hergent, and H.G. Zimmermann, "Improving model selection by non-convergent methods," *Neural Networks*, vol. 6, pp. 771–783, 1993.

[18] L.J. Fogel, A.J. Owens, and M.J. Walsh, *Artificial Intelligence Through Simulated Evolution*. New York: Wiley, 1966.

[19] G.B. Fogel and D.B. Fogel, "Continuous evolutionary programming: Analysis and experiments," *Cybernet. Syst.*, vol. 26, pp. 79–90, 1995.

[20] D.B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. New York: IEEE Press, 1995.

[21] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[22] P.J.B. Hancock, "Genetic algorithms and permutation problems: a comparison of recombination operators for neural net structure specification," *Proc. Int. Workshop Combinations Genetic Algorithms Neural Networks (COGANN-92)*, D. Whitley and J. D. Schaffer, Eds., Los Alamitos, CA: IEEE Comput. Soc. Press, pp. 108–122, 1992.

[23] L.K. Hansen and P. Salamon, "Neural network ensembles," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, no. 10, pp. 993–1001, 1990.

[24] S. Hashem, "Optimal linear combinations of neural networks," Ph.D. dissertation, School Ind. Eng., Purdue Univ., West Lafayette, IN, Dec. 1993.

[25] Kalyanmoy Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*, Chichester, UK: Wiley, 2001.

[26] V. Khare, X. Yao, and B. Sendhoff, "Multi-network evolutionary systems and automatic problem decomposition," *International Journal of General Systems*, vol. 35, no. 3, pp. 259–274, 2006.

[27] A. Krogh and J. Vedelsby, "Neural network ensembles, cross validation, and active learning," *Neural Inf. Process. Syst.*, vol. 7, pp. 231–238, 1995.

[28] A. Krogh and P. Sollich, "Statistical mechanics of ensemble learning," *Phys. Rev. E*, vol. 55, pp. 811–825, 1997.

[29] T.Y. Kwok and D.Y. Yeung, "Constructive algorithms for structure learning in feed-

forward neural networks for regression problems," *IEEE Trans. Neural Networks*, vol. 8, pp. 630–645, May 1997.

[30] T.Y. Kwok and D.Y. Yeung, "Objective functions for training new hidden units in constructive neural networks," *IEEE Trans. Neural Networks*, vol. 8, pp. 1131–1148, Sept. 1997.

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

[32] C.Y. Lee and X. Yao, "Evolutionary programming using the mutations based on the Lévy probability distribution," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 1, pp. 1–13, 2004.

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

[34] Y. Liu, X. Yao, and T. Higuchi, "Evolutionary ensembles with negative correlation learning," *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 4, pp. 380–387, 2000.

[35] J. MacQueen, "Some methods for classification and analysis of multivariate observation," in *Proc. 5th Berkeley Symp. Mathematical Statistics and Probability*, Berkeley, CA: University of California Press, vol. 1, pp. 281–297, 1967.

[36] S.W. Mahfoud, "Niching methods for genetic algorithms," Ph.D.thesis, Dept. General Engineering, Univ. Illinois, Urbana-Champaign, IL, 1995.

[37] Md. Monirul Islam, X. Yao, and K. Murase, "A constructive algorithm for training cooperative neural network ensembles," *IEEE Transactions on Neural Networks*, vol. 14, pp. 820–834, 2003.

[38] B. Mulgrew and C.F.N. Cowan, *Adaptive Filters and Equalizers*. Boston, MA: Kluwer, 1988.

[39] S.V. Odrí, D.P. Petrovacki, and G.A. Krstonosic, "Evolutional development of a multi-level neural network," *Neural Networks*, vol. 6, no. 4, pp. 583–595, 1993.

[40] D.W. Opitz and J.W. Shavlik, "Generating accurate and diverse members of a neural-network ensemble," *Neural Inf. Process. Syst.*, vol. 8, pp. 535–541, 1996.

[41] D. Opitz and R. Maclin, "Popular ensemble methods: an empirical study," *J. Artif. Intell. Res.*, vol. 11, pp. 169–198, 1999.

[42] M.P. Perrone, "Improving regression estimation: Averaging methods for variance reduction with extensions to general convex measure optimization," Ph.D. dissertation, Dept. Physics, Brown Univ., Providence, RI, May 1993.

[43] L. Prechelt, "Proben1-A set of neural network benchmark problems and benchmarking rules," *Tech. Rep. 21/94*, Fakultät für Informatik, Univ. Karlsruhe, Karlsruhe, Germany, Sept. 1994.

[44] L. Prechelt, "Some notes on neural learning algorithm benchmarking," *Neurocomputing*, vol. 9, no. 3, pp. 343–347, 1995.

[45] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. 14, pp. 465–471, 1978.

[46] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning internal representations by error propagation," in D. E. Rumelhart & J. L. McClelland (ed.), *Parallel Distributed Processing: Explorations in the Microstructures of Cognition, I*, Cambridge, MA: MIT Press, pp. 318–362, 1986.

[47] A.J.C. Sharkey, "On combining artificial neural nets," *Connect. Sci.*, vol. 8, no. 3/4, pp. 299–313, 1996.

[48] J.D. Schaffer, D. Whitley, and L.J. Eshelman, "Combinations of genetic algorithms and neural networks: A survey of the state of the art," in *Proc. Int. Workshop Combinations Genetic Algorithms Neural Networks (COGANN-92)*, D. Whitley and J. D. Schaffer (Eds.), Los Alamitos, CA: IEEE Comput. Soc. Press, pp. 1–37, 1992.

[49] N. Srinivas and K. Deb, "Multi-objective function optimization using non-dominated sorting genetic algorithms," *Evolutionary Computation*, vol. 2, no. 3, pp. 221–248, 1994.

[50] R. Storn and K. Price, "Minimizing the real functions of the ICEC'96 contest by differential evolution," *Proc. of the IEEE International Conference on Evolutionary Computation*, Nagoya, pp. 842–844, May 1996.

[51] E.K. Tang, P.N. Suganthan, and X. Yao, "An Analysis of Diversity Measures," *Machine Learning*, vol. 65, pp. 247–271, 2006.

[52] X. Yao, "Evolution of connectionist networks," *Proc. of the Int. Symp. AI, Reasoning & Creativity*, Queensland, Australia, Griffith Univ., pp. 49–52, 1991.

[53] X. Yao, "An empirical study of genetic operators in genetic algorithms," *Microprocess. Microprog.*, vol. 38, pp. 707–714, 1993.

[54] X. Yao, "A review of evolutionary artificial neural networks," *Int. J. Intell. Syst.*, vol. 8, no. 4, pp. 539–567, 1993.

[55] X. Yao, "Evolutionary artificial neural networks," *Int. J. Neural Syst.*, vol. 4, no. 3, pp. 203–222, 1993.

[56] X. Yao, "The evolution of connectionist networks," in *Artificial Intelligence and Creativity*, T. Dartnall (Ed.), Dordrecht, The Netherlands: Kluwer, pp. 233–243, 1994.

[57] X. Yao, "Evolutionary artificial neural networks," in *Encyclopedia of Computer Science and Technology*, vol. 33, A. Kent and J. G. Williams (Eds.), New York: Marcel Dekker, pp. 137–170, 1995.

[58] X. Yao and Y. Shi, "A preliminary study on designing artificial neural networks using co-evolution," in *Proc. IEEE Singapore Int. Conf. Intelligent Control Instrumentation*, pp. 149–154, June 1995.

[59] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, pp. 1423–1447, 1999.

[60] X. Yao and Y. Liu, "Ensemble structure of evolutionary artificial neural networks," in *Proc. of 1996 IEEE Int. Conf. Evolutionary Computation (ICEC'96)*, Nagoya, Japan, pp. 659–664, 1996.

[61] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Trans. Neural Networks*, vol. 8, no. 3, pp. 694–713, 1997.

[62] X. Yao and Y. Liu, "Making use of population information in evolutionary artificial neural networks," *IEEE Trans. Syst., Man, Cybern., B*, vol. 28, no. 3, pp. 417–425, 1998.

[63] X. Yao, Y. Liu and P. Darwen, "How to make best use of evolutionary learning," in *Complex Systems: From Local Interactions to Global Phenomena*, R. Stocker, H. Jelinek, and B. Dumortier (Eds.), Amsterdam, Netherlands: IOS Press, pp. 229–242, 1996.

[64] X. Yao, Y. Liu and G. Lin, "Evolutionary Programming Made Faster," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, pp. 82–102, 1999.