Evolving Artificial Neural Network Ensembles

Md. Monirul Islam[1] and Xin Yao[2]

[1] Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka 1000, Bangladesh*, mdmonirulislam@cse.buet.ac.bd

[2] Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA), School of Computer Science, University of Birmingham, Edgbaston, Birmingham B15 2TT, UK, x.yao@cs.bham.ac.uk

1 Introduction

Artificial neural networks (ANNs) and evolutionary algorithms (EAs) are both abstractions of natural processes. In the mid 1990s, they were combined into a computational model in order to utilize the learning power of ANNs and adaptive capabilities of EAs. Evolutionary ANNs (EANNs) is the outcome of such a model. They refer to a special class of ANNs in which evolution is another fundamental form of adaptation in addition to learning [52–57]. The essence of EANNs is their adaptability to a dynamic environment. The two forms of adaptation in EANNs – namely evolution and learning – make their adaptation to a dynamic environment much more effective and efficient. In a broader sense, EANNs can be regarded as a general framework for adaptive systems – in other words, systems that can change their architectures and learning rules appropriately without human intervention.

EAs have been introduced into ANNs at roughly three different levels: (i) connection weights, (ii) architectures, and (iii) learning rules. The evolution of connection weights introduces an adaptive and global approach to training, especially in the reinforcement learning and recurrent network learning paradigms, where gradient-based training algorithms often experience great difficulties. Architecture evolution enables ANNs to adapt their topologies to different tasks without human intervention. 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.

There is strong biological and engineering evidence to support the assertion that the information processing capability of ANNs is determined by their

* Portions reprinted with permission, from X. Yao and M.M. Islam, "Evolving artificial neural network ensembles," IEEE Computational Intelligence Magazine, 3(1):31–42, February 2008. Copyright IEEE.

M.M. Islam and X. Yao: *Evolving Artificial Neural Network Ensembles*, Studies in Computational Intelligence (SCI) **115**, 851–880 (2008)
www.springerlink.com

architecture. A large amount of the literature is therefore devoted to finding optimal or near optimal ANN architectures by using EAs (see review papers [48,54,59]). However, many real-world problems are too large and too complex for a single ANN alone to solve. There are ample examples from both natural and artificial systems that show that an integrated system consisting of several subsystems can reduce the total system complexity while satisfactorily solving a difficult problem. Many successes in evolutionary computation have already demonstrated this. A typical example of the success of ANN ensembles in improving classifier generalization is [62].

ANN ensembles adopt the divide-and-conquer strategy. Instead of using a single network to solve a task, an ANN ensemble combines a set of ANNs that learn to subdivide the task and thereby solve it more efficiently and elegantly. It offers several advantages over a monolithic ANN [47]. First, it can perform more complex tasks than any of its components (that is, individual ANNs in the ensemble). Second, it can make an 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 which show that ensembles, if designed appropriately, usually perform (generalize) better than any single member system. A theoretical account of why ensembles perform better than single learners is presented in [12]. 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 predictors which perform better than random guessing and produce uncorrelated outputs. This means individual ANNs in the ensemble need to be accurate as well as diverse (also mentioned in one of the seminal works by Hansen and Salamon [23]). Krogh and Sollich formally show that an ideal ensemble is one that consists of highly correct (accurate) predictors which at the same time disagree – in other words, uncorrelate as much as possible (that is, substantial diversity amongst members is exhibited) [28]. This has also been tested and empirically verified [40,41]. Given that ANN ensembles generalize better as compared with a single ANN, ensemble research has become an active research area and has seen an influx of researchers devising myriad algorithms trying to improve the prediction ability of such aggregate systems in recent years.

2 Evolutionary Ensembles

Although there have been many studies on how to evolve ANNs more effectively and efficiently [48, 54, 59] the issue of how to form the final result from an evolutionary process has been overlooked [60]. The best individual in the last generation or among all the generations is generally considered as the final

result. However, the best individual (that is, ANN) with respect to training or validation data may not be the best for unseen testing data. The remaining individuals in the population may contain some useful information that may improve the generalization performance of ANNs.

The aim of this Section is to present an approach proposed by [62] to form the final result of an evolutionary process. Unlike most previous work, the approach utilizes the population information, rather than an individual's information, to form the final result. It considers each individual in a population as a module. Thus different individuals in the last generation are linearly combined by regarding a population of ANNs as an *ensemble*. The reason for using a linear combination is its simplicity, although non-linear combination methods could be used. The idea of combining different modules is not new and has been studied in both the ANN field and statistics [24, 42]. However, few attempts have been made in evolutionary learning to use population information in forming the final system.

The proposed approach was applied on three real-world problems to demonstrate the effectiveness of using the population information in forming the final result of an evolutionary process. `EPNet` [61] was used to evolve a population of ANNs. Four linear combination methods were used to form the final result. They were majority voting, the rank-based linear combination method, the recursive least-square (RLS) algorithm [38], and the subset method.

2.1 An Evolutionary Design System for ANNs – EPNet

`EPNet` [61] is an automatic ANN design algorithm based on evolutionary programming (EP) [18, 20]. It uses an EP algorithm for evolving ANN architectures and a hybrid training scheme for learning their connection weights. It puts the emphasis on evolving ANN behaviors, which is the main reason for using EP in its evolutionary scheme. Since `EPNet` evolves architectures and learns connection weights of ANNs simultaneously, it reduces the noise in fitness evaluation [61] unlike some previous studies (for instance, [58]). The main structure of `EPNet` is shown in Fig. 1; a detailed description can be found in [61].

`EPNet` relies on novel mutations and a rank-based selection scheme [53]. It does not use recombination operators in order to avoid the permutation problem (that is, competing conventions) [6, 9, 22]. This not only makes the evolution inefficient, but also makes crossover operators more difficult to produce highly fit offspring. To determine an ANN architecture for a given problem, an algorithm needs to select the number of hidden nodes and connections required for the ANN in solving the problem. In addition, it needs to assign weights for the architecture. `EPNet` uses five mutations one after another to achieve these goals. If one mutation is successful then other mutations are

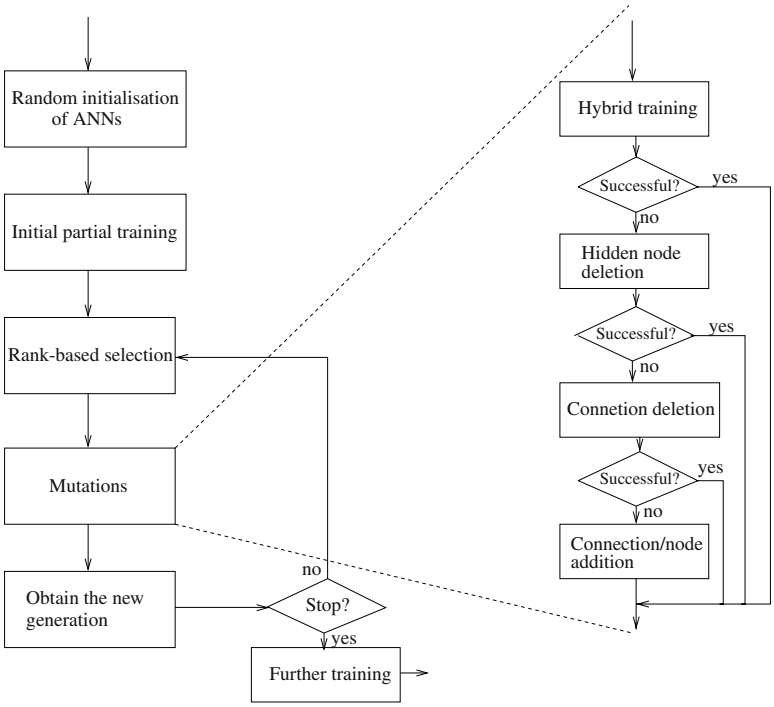

Fig. 1. The major steps of `EPNet` [61] © 1997

not applied. The mutations used in `EPNet` are hybrid training, node deletion, connection deletion, connection addition, and node addition. Each mutation operator in `EPNet` produces one offspring that can replace at most one individual in any generation of a population. This replacement strategy is very similar to the one used in a steady-state GA [51], or in a continuous EP [19]. The advantage of such replacement has been demonstrated in [19, 51].

The hybrid training scheme consists of 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 minima problem of the BP algorithm. A distinct feature of hybrid training is that it is partial. This means that `EPNet` does not train each individual until it converges rather it trains the individual for a fixed number of epochs in each generation. The number of epochs is a user specified parameter. The main reason behind partial training is to increase the computational efficiency in fitness evaluation. Hybrid training is always attempted before any architectural mutations (namely, node/connection deletion/addition) because the latter cause larger changes in ANN behavior.

Node deletion in `EPNet` is done totally at random – in other words, a node is selected uniformly at random for deletion. However, the other three

architectural mutations are not uniformly 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 [17,61]. Node addition is achieved by splitting an existing node [39], rather than by introducing a random one. The 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:

$$\begin{aligned} w_{ij}^1 &= w_{ij}^2 = w_{ij}, & i \geq j \\ w_{ki}^1 &= (1+\alpha) w_{ki} & i < k \\ w_{ki}^2 &= -\alpha w_{ki} & i < k \end{aligned} \tag{1}$$

where $\boldsymbol{w}$ is the weight vector of the existing node i, $\boldsymbol{w}^1$ and $\boldsymbol{w}^2$ 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. This method helps greatly in maintaining the behavioral link between the parent and its offspring. It also reduces blindness caused by a random node.

To improve the generalization ability of evolved ANNs, validation sets are used in `EPNet`. Each individual (that is, ANN) is trained on a training set, but it is evaluated on a validation set. All fitness values are calculated based on the validation, 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 Combination Methods

The four combining methods used in `EPNet` are all linear. The simplest linear combination method is majority voting. That is, the output of the most number of EANNs will be the output of the ensemble. If there is a tie, the output of the EANN (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 weight for each individual. The rank-based linear combination method is such a scheme that puts weight on each ANN in the population based on their fitness values. More specifically, we can use rankings to generate weights for each EANN in combining the ensemble output. That is, given N sorted EANNs

with an increasing error rate, where N is the population size, and their outputs $o_1, o_2, \cdots, o_N$, then the weight for the ith EANN is:

$$w_i = \frac{exp(\beta(N+1-i))}{\sum_{j=1}^{N} exp(\beta_j)} \tag{2}$$

where β is a scaling factor. The ensemble output is:

$$O = \sum_{j=1}^{N} w_j o_j. \tag{3}$$

One of the well-known algorithms for learning linear combination weights (that is, 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. The detailed description of the RLS algorithm implemented here can be found in [38].

In the above three combination methods, all the individuals in the last generation were used in forming ensembles. It is interesting to investigate whether one can reduce the size of the ensembles without too much increase in testing error rates. Such investigation can provide some hints on whether all the individuals in the last generation will contain some useful information and shed some light on the importance of a population in evolutionary learning. As the space of possible subsets is very large ($2^N - 1$) for a population of size N, it is impractical to use exhaustive search to find an optimal subset. Instead, a genetic algorithm (GA) [21] is used to search for a near-optimal subset [62]. The weights for each EANN in each subset were determined by the same RLS algorithm [38] as used in the previous scheme.

2.3 Experimental Studies

`EPNet` was applied on three real-world problems. 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 the Australian credit card data set. The problem is to assess applications for a credit card based on a number of attributes; the 14 attributes include six numeric values and eight discrete ones. The output has two classes.

The diabetes data is also a two class problem. 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 attributes. The aim of 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 data set of the heart problem has 13 attributes, which

have been extracted from a larger set of 75. The description of the extraction process can be found in [62].

Two validation sets were used in all experiments. One validation set, V-set 1, was used in the fitness evaluation. The other validation set, V-set 2, was used in further training of `EPNet`. The best individual with the minimum error rate on V-set 2 was chosen as the final result. 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.

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 were 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 re-scaled 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 the ANNs. In this method, the output with the highest activation designates the class.

The same parameters were used for all data sets. These were as follows: population size (20); maximum number of generations (100); 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); initial connection density (0.75, which means the probability of having a connection between two nodes is 75%; the constraint of feedforward ANNs cannot be violated of course); 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` was not very sensitive to these parameters.

Results

Table 1 summarizes the results of [62]. The best individual in the last generation and the ensemble formed by the 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 differences among 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

Table 1. Testing accuracies of the best individual in a population and ensemble formed from individuals in the population by using majority voting, the RLS algorithm [38] and 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 this table have been summarized from [62])

Problem		Best individual	Rank-based	Error rate majority voting	RLS algorithm	Optimal subset
Credit 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

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 is clear from Table 1 that the results of the ensemble formed by the rank-based linear method are either better than or as good as those produced by 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 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 levels 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 search time.

All the above results indicate that a population contains more information than any individual in it. Such information can be used effectively to improve generalization of the learning systems. 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. However, no special considerations were made in the evolution of ANNs about modularization in `EPNet`. If the evolution of modular ANNs could be encouraged in the evolutionary process, one can expect to improve the results further. One way to encourage modularization is by speciation. That is, we can use techniques like fitness sharing [14,21] to automatically form species in a population. Each species will be a specialist in dealing with part of a complex problem and will be treated as a module of the final system. In this case, modules are evolved specifically for an integrated system. Co-evolutionary learning is usually used in evolving modular systems. This idea has been tested successfully in a rule-based system [15] and described in the next Section.

3 Automatic Modularization

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

The main idea behind automatic modularization 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 subsystem (module) in the integrated system, which is represented by the entire population. The experimental results have shown that automatic modularization can lead to substantial performance gain in evolving game-playing strategies for iterated prisoner's dilemma games [15].

Although the original work on automatic modularization was done using rule-based systems, the idea is equally applicable to neural networks, decision trees and other classifiers. [26] described the most recent work related to automatic modularization using neural networks.

4 Negative Correlation Learning

Although ANN ensembles perform better than single ANN in many cases, a number of issues 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 negatively correlated, 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 and are not accurate.

Liu and Yao 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 ensembles [33]. A number of works (for example, [13, 34, 37]) have utilized this feature in training ensembles. 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 networks to encourage specialization and cooperation among the individual ANNs.

Suppose that we have a training set

$$D = \{(\mathbf{x}(1), d(1)), \cdots, (\mathbf{x}(N), d(N))\} \tag{4}$$

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. This Section considers estimating d by forming an ensemble whose output is a simple averaging of outputs of a set of ANNs

$$F(n) = \frac{1}{M} \Sigma_{i=1}^{M} F_i(n) \tag{5}$$

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

NCL introduces a correlation penalty term into the error function of each individual network in the ensemble so that all the networks 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

$$\begin{aligned} E_i &= \frac{1}{N}\Sigma_{n=1}^{N} E_i(n) \\ &= \frac{1}{N}\Sigma_{n=1}^{N}\frac{1}{2}(F_i(n) - d(n))^2 + \frac{1}{N}\Sigma_{n=1}^{N}\lambda p_i(n) \end{aligned} \tag{6}$$

where $E_i(n)$ is the value of the error function of network i at presentation of the nth training pattern. The first term in the right side of Eqn. (6) is the empirical risk function of network i. The second term, p_i, is a correlation penalty function. The purpose of minimizing p_i is to negatively correlate each network'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 has the form:

$$p_i(n) = (F_i(n) - F(n))\,\Sigma_{j\neq i}\,(F_j(n) - F(n)) \tag{7}$$

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

$$\begin{aligned} \frac{\partial E_i(n)}{\partial F_i(n)} &= F_i(n) - d(n) + \lambda\frac{\partial p_i(n)}{\partial F_i(n)} \\ &= F_i(n) - d(n) + \lambda\Sigma_{j\neq i}\,(F_j(n) - F(n)) \\ &= F_i(n) - d(n) - \lambda(F_i(n) - F(n)) \\ &= (1-\lambda)(F_i(n) - d(n)) + \lambda(F(n) - d(n)) \end{aligned} \tag{8}$$

where we have made use of the assumption that $F(n)$ has constant value with respect to $F_i(n)$. The standard BP algorithm [46] has been used for weight adjustments in the mode of pattern-by-pattern updating. That is, weight updating of all the individual networks is performed simultaneously using Eqn. (8) after the presentation of each training pattern. One complete presentation of the entire training set during the learning process is called an 'epoch'.

NCL from Eqn. (8) 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 ith network. From Eqns. (6)–(8), we may 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 Eqn. (8) we get

$$\frac{\partial E_i(n)}{\partial F_i(n)} = F(n) - d(n) \tag{9}$$

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

$$E_{ens}(n) = \frac{1}{2}\left(\frac{1}{M}\Sigma_{i=1}^{M}F_i(n) - d(n)\right)^2 \tag{10}$$

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

$$\begin{aligned}\frac{\partial E_{ens}(n)}{\partial F_i(n)} &= \frac{1}{M}\left(\frac{1}{M}\Sigma_{i=1}^{M}F_i(n) - d(n)\right) \\ &= \frac{1}{M}(F(n) - d(n))\end{aligned} \tag{11}$$

In this case, we get

$$\frac{\partial E_i(n)}{\partial F_i(n)} \propto \frac{\partial E_{ens}(n)}{\partial F_i(n)} \tag{12}$$

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 and Yao proposed evolutionary ensembles with NCL (EENCL) to determine automatically the number of individual ANNs in an ensemble and to exploit the interaction between individual ANN design and their combination [34]. In EENCL, an evolutionary algorithm based on EP [18, 20] was used to search for a population of diverse individual ANNs for constructing ensembles. This means an evolutionary algorithm is used here for determining automatically the number of ANNs required for constructing an ensemble.

Two schemes are used in EENCL to maintain diversity in different individuals (that is, ANNs) of the population. They are fitness sharing [63] and

NCL [33]. The fitness sharing accomplishes speciation by degrading the raw fitness (in other words, 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 over all training examples.

EENCL uses Gaussian mutation to produce offspring from parents, although non-Gaussian mutation such as Cauchy mutation [64] and Lévy mutation [32] can also be used. The mutation is carried out in two steps: (i) weight mutation, and (ii) further weight training. In the first step, n_b parent networks are selected at random to create n_b offspring. The parameter n_b is a parameter specified by a user. The probability for selecting a parent network is same. The following is used for weight mutation [34]:

$$w_{ij}^{'} = w_{ij} + N(0,1) \tag{13}$$

where $w_{ij}^{'}$ and w_{ij} denote the weights of offspring i and parent i, respectively, $i = 1, \cdots, n_b$, 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 has finished. Now a question arises as to how to form the ensemble from a population of ANNs. The most convenient way is to use *all* ANNs – that is, the whole population in the last generation. The other way is to use a subset of 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 (such as the k-means algorithm) [35]. The latter may reduce the size of an ensemble without worsening its performance too much. In EENCL, these two approaches were used to form ensembles.

Three combination methods were used to determine the output of an ensemble from different ANNs used for forming the ensemble, these being 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 greatest number of individual 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.

4.2 Experimental Studies

EENCL was applied on two benchmark problems: the Australian credit card assessment 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.

The same parameters were used for both problems. They are as follows: population size 25, number of generations 200, reproduction block size n_b 2, strength parameter λ for NCL [33] 0.75, number of training epochs 5, minimum number of cluster sets 3, and the maximum number of cluster sets 25. The ANNs used in the population are multilayer perceptrons with one hidden layer and five hidden nodes.

Results

All the results presented in this Section are summarized from results presented in [34]. Table 2 shows the results of EENCL for the two data sets, where the

Table 2. Accuracy rates of EENCL for the Australian credit card and the diabetes data sets. The results are averaged on 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 (Note that the results presented in this table have been summarized from [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

Table 3. Accuracy rates of the ensemble formed by the representatives from species. The results are averaged on 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 (Note that the results presented in this table have been summarized from [34])

Accuracy rate	Credit 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

ensembles were formed using the whole population in the last generation. The *accuracy rate* refers to the percentage of correct classifications produced by EENCL. Comparing the accuracy rates obtained by the three combination methods, winner-takes-all outperformed simple averaging and majority voting on both problems. In simple averaging and majority voting, all individuals are treated equally. However, not all individuals are equally important. Because different individuals created by EENCL were able to specialize to different parts of the testing set, only the outputs of these specialists should be considered to make the final decision of the ensemble for this part of the testing set. The winner-takes-all combination method performed better because there are good and poor individuals for each pattern in the testing set, and winner-takes-all selects the best individual.

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

Determination of ensemble size by an evolutionary approach was presented in Sect. 4.1. The problem with ENNCL [34] is that it only determines the number of individual ANNs in the ensemble automatically, but the sizes of the ANNs

Table 4. Sizes of the ensembles using the representatives from species. The results are averaged on 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 (Note that the results presented in this table have been summarized from [34])

	Ensemble size			
	Mean	SD	Min	Max
Credit card	13.2	7.8	5	25
Diabetes	16.3	6.4	5	25

need to be specified by the user. It is well known that the accuracy of ANNs is greatly dependent on their size. This means random selection of the ANN sizes may hurt the ensemble performance. This is because the performance of ensembles not only depends on the diversity of individual ANNs but also on ANN accuracy. The aim of this Section is to present a constructive algorithm for training cooperative neural-network ensembles (CNNEs) [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 the 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 epochs, which is determined automatically by its training process. Like 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 CNNE's 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 the ANNs, and the number of epochs required for training ANNs – need to be addressed when designing ensembles. This means 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.

The main structure of CNNE is shown in Fig. 2, and a detailed description can be found in [37]. It is not clear from the figure when and how to add hidden nodes and individual ANNs to the ensemble architecture. CNNE uses

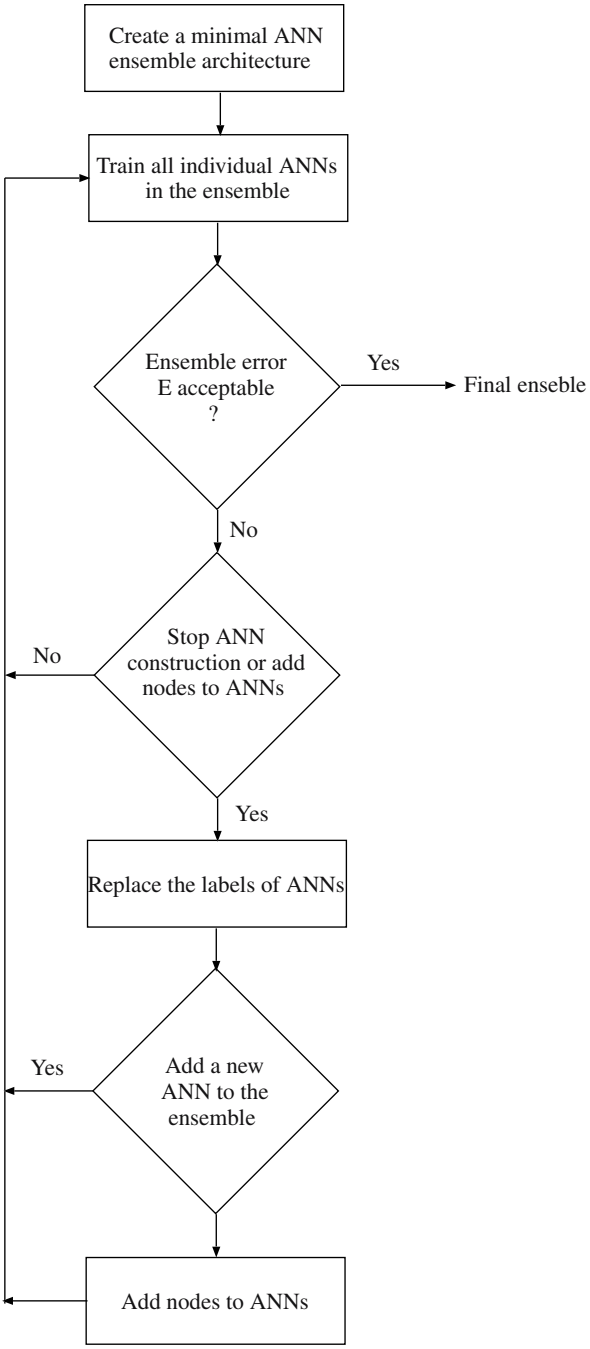

Fig. 2. The major steps of CNNE [37] © 2003

a simple criteria for adding hidden nodes and ANNs, based on the contribution of ANNs to the ensemble. The following is used to determine the contribution:

$$C_i = 100\left(\frac{1}{E} - \frac{1}{E^i}\right) \tag{14}$$

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 ANN 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

CNEE was applied to seven benchmark problems: the Australian credit card assessment problem, the breast cancer problem, the diabetes problem, the glass problem, the heart disease problem, the letter recognition problem, and the soybean problem. The data sets 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 16,000 and 2,000 examples were randomly selected from 20,000 examples for the training and validation sets, and the remaining 2,000 examples were used for the testing set.

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

Results

Table 5 show the results of CNNE over 30 independent runs on the seven different problems. The results presented in this table are summarized from [37]. The error rates in the table refer to the percentage of wrong classifications

Table 5. Architectures and accuracies of ensembles produced by CNNE for seven different 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 (Note that the results presented in this table have been summarized from [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
Credit card	6.5	5.3	0.090	7.8	4.7	0.092	7.4	4.3	0.091
Breast 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

produced by the trained ensemble on the testing set. M and N represent the number of ANNs in an ensemble and of hidden nodes in an ANN, respectively.

It can be observed 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 credit 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, and the average number of individual ANNs and hidden nodes were 7.8 and 4.7, respectively, when $\tau = 10$ and $m_h = 2$. This indicates that for the same value of τ the number of individual ANNs in an ensemble increases when the number of hidden nodes in the 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.

The ability of CNNEs to automatically construct different ensembles for different problems can be clearly seen from Table 5. CNNE produced large ensembles for the letter problem, which is large in comparison with the other problems here, and smaller ensembles for other problems. However, there are other factors in addition to the size of training sets – for example the complexity of the given problem and noise in the training set – that influence the ensemble architecture. For instance, the number of training examples for the diabetes problem was 384, while it was 342 for the soybean problem. In terms of average results, CNNE produced ensembles that had 4.7 individual ANNs with 4.5 hidden nodes for the diabetes problem, while it produced ensembles

that had 5.3 individual ANNs with 5.5 hidden nodes for the soybean problem. In general, all the above examples illustrated the same point, namely CNNEs ability to determine the ensemble automatically for different problems without human intervention.

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 [3], in which a new algorithm, called memetic Pareto artificial neural network (MPANN) is proposed for training ANNs. It combines a multi-objective evolutionary algorithm and a gradient-based local search in reducing network complexity and training error. MPANN was later applied for 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 we are getting a set of near optimal ANNs that can easily be used for constructing ensembles. In addition, the use of an evolutionary approach would speed up finding a set of near optimal solutions. This is because the evolutionary approach uses a multi-directional search scheme instead of a unidirectional search scheme as used by conventional approaches.

Recently Chandra and Yao proposed an algorithm, called diverse and accurate ensemble learning algorithm (DIVACE), that a uses multi-objective evolutionary approach to ensemble learning [13]. DIVACE tries to find an optimum tradeoff between diversity and accuracy by treating them explicitly as multi-evolutionary pressures. [12] give a good account of why diversity is necessary in ANN ensembles, and present a taxonomy of methods that enforce it and which are used in practice. The evolutionary process of DIVACE is quite similar to the one used in pareto differential evolution [1] and in MPANN [4,5]. It also has three main components – namely fitness evaluation, selection and genetic operations, as with conventional EAs.

Fitness evaluation in DIVACE is not straightforward and is based on the non-dominated sorting procedure proposed by [49]. This sorting procedure can be described as follows: Let there are two solutions S_1 and S_2 for a given problem. A solution is considered optimal if it satisfies all n objective of the problem. According to [49], the solution S_1 is said to 'dominate' solution S_2 if S_1 is not worse than S_2 in all n objectives and S_1 is strictly better than S_2 in at least one objective. This is the concept of non-domination. After non-dominated sorting, a set of individuals is found which is better than the rest of the individuals in the population.

Since the evolutionary process of DIVACE is similar to one used in Pareto differential evolution [1] – a variant of differential evolution [50] – three parents are randomly selected from the non-dominated set for the crossover and mutation genetic operations. DIVACE incorporates the idea of simulated annealing, which makes the variance of the Gaussian distribution used for crossover adaptive. The following equations are used to produce offspring by crossover:

$$w_{hi} = w_{hi}^{\alpha_1} + N\left(0, \sigma^2\right)\left(w_{hi}^{\alpha_2} - w_{hi}^{\alpha_3}\right) \tag{15}$$

$$w_{oh} = w_{oh}^{\alpha_1} + N\left(0, \sigma^2\right)\left(w_{oh}^{\alpha_2} - w_{oh}^{\alpha_3}\right) \tag{16}$$

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 on an offspring generated by crossover with probability $1/N$, where N is the size of the population. The following equations are used for mutation.

$$w_{hi} = w_{hi} + N\left(0, 0.1\right) \tag{17}$$

$$w_{oh} = w_{oh} + N\left(0, 0.1\right) \tag{18}$$

6.1 Experimental Studies

This Section presents some results obtained on testing DIVACE on the Australian credit card assessment and diabetes problems. 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 card and diabetes problems, respectively. Three combining methods – namely simple averaging, majority voting and winner-takes-all are used in the experiments.

Results

During the course of the evolutionary process, it was expected that each member in the Pareto (non-dominated) set (after every generation) would perform well on different parts of the training set. Table 6 shows the performance

Table 6. Performance (accuracy rates) of the ensemble formed using DIVACE on the Australian credit card assessment data set (the results in this table are summarized from [13])

	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 data set (the results in this table are summarized from [13])

	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

accuracy of the formed ensemble on the Australian credit card assessment data set. Table 7 shows the same for the Diabetes data set. Good performance can be observed for the DIVACE algorithm.

7 Conclusions

Combining ANNs with evolutionary computation has been a popular topic since the late 1980s. While the early work tended to focus on evolving single ANNs, 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 becomes larger and more complex; a 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. This Chapter has reviewed some of the latest developments in the area of evolving ANN ensembles.

Acknowledgements

Portions of this chapter originally appeared in X. Yao and Md. M. Islam, Evolving artificial neural network ensembles, IEEE Computational Intelligence Magazine, 3(1): 31–42, February 2008. Permission to reprint this material in the current Compendium is gratefully acknowledged.

References

1. Abbass HA, Sarker R, Newton C (2001) PDE: A Pareto-frontier differential evolution approach for multi-objective optimization problems. In: Kim J-H (ed.) *Proc. IEEE Conf. Evolutionary Computation (CEC2001)*, 27–30 May, Seoul, South Korea. IEEE Press, Piscataway, NJ: 971–978.

2. Abbass HA (2002) The self-adaptive Pareto differential evolution algorithm. In: Fogel DB, El-Sharkawi MA, Yao X, Greenwood G, Iba H, Marrow P, Shackleton M (eds.) *Proc. IEEE Conf. Evolutionary Computation (CEC2002)*, 12–17 May, Honolulu, HI. IEEE Press, Piscataway, NJ: 831–836.
3. Abbass HA (2003) Speeding up backpropagation using multiobjective evolutionary algorithms. *Neural Computation*, 15(11): 2705–2726.
4. Abbass HA (2003) Pareto neuro-evolution: constructing ensemble of neural networks using multi-objective optimization. In: Sarker R, Reynolds R, Abbass H, Tan KC, McKay B, Essam D, Gedeon T (eds.) *Proc. IEEE Conf. Evolutionary Computation (CEC2003)*, 8–12 December, Canberra, Australia. IEEE Press, Pisctaway, NJ: 2074–2080.
5. Abbass HA (2003) Pareto neuro-ensemble. In: Gedeon TD, Chun L, Fung C (eds.) *Proc. 16th Australian Joint Conf. Artificial Intelligence*, 3–5 December, Perth, Australia. Springer-Verlag, Berlin: 554–566.
6. Angeline PJ, Sauders GM, Pollack JB (1994) An evolutionary algorithm that constructs recurrent neural networks. *IEEE Trans. Neural Networks*, 5(1): 54-65.
7. Baldi PF, Hornik K (1995) Learning in linear neural networks: a survey. *IEEE Trans. Neural Networks*, 6(4): 837–858.
8. Blake C, Merz C *UCI repository of machine learning databases.* (available online at http://www.ics.uci.edu/m̃learn/MLRepository.html – last accessed September 2007).
9. Belew RK, McInerney J, Schraudolph NN (1991) Evolving networks: using genetic algorithm with connectionist learning. *Technical Report CS90- 174 (revised)*, Computer Science and Engineering Department (C-014), University of California, San Diego, February.
10. Bollé D, Dominguez DRC, Amari S (2000) Mutual information of sparsely coded associative memory with self-control and tenary neurons. *Neural Networks*, 1: 452–462.
11. Brown G, Wyatt JL (2003) Negative correlation learning and the ambiguity family of ensemble methods. In: Windeatt T, Roli F (eds.) *Proc. Intl. Workshop Multiple Classifier Systems*, 11–13 June, Guildford, UK. Springer-Verlag, Berlin: 266–275.
12. Brown G, Wyatt JL, Harris R, Yao X (2005) Diversity creation methods: a survey and categorisation. *J. Information Fusion,* 6: 5–20.
13. Chandra A, Yao X (2006) Ensemble learning using multi-objective evolutionary algorithms. *J. Mathematical Modeling and Algorithms*, 5(4): 417–445.
14. Darwen PJ, Yao X (1996) Every niching method has its niche: fitness sharing and implicit sharing compared. In: Ebeling W, Rechenberg I, Schwefel H-P, Voight H-M (eds.) *Parallel Problem Solving from Nature (PPSN) IV*, 22–26 September, Berlin, Germany. Lecture Notes in Computer Science 1141. Springer-Verlag, Berlin: 398–407.
15. Darwen PJ, Yao X (1997) Speciation as automatic categorical modularization. *IEEE Trans. Evolutionary Computation*, 1: 101–108.
16. Dietterich TG (1998) Machine-learning research: four current directions. *AI Magazine*, 18(4): 97–136.
17. Finnoff W, Hergent F, Zimmermann HG (1993) Improving model selection by nonconvergent methods. *Neural Networks*, 6: 771–783.
18. Fogel LJ, Owens AJ, Walsh MJ (1966) *Artificial Intelligence Through Simulated Evolution.* Wiley, New York, NY.

19. Fogel GB, Fogel DB (1995) Continuous evolutionary programming: analysis and experiments. *Cybernetic Systems*, 26: 79–90.
20. Fogel DB (1995) *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway, NJ.
21. Goldberg DE (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA.
22. Hancock PJB (1992) Genetic algorithms and permutation problems: a comparison of recombination operators for neural net structure specification. In: Whitley D, and Schaffer JD (eds.) *in Proc. Intl. Workshop Combinations Genetic Algorithms Neural Networks (COGANN-92)*, 6 June, Maryland, IEEE Computer Society Press, Los Alamitos, CA: 108–122.
23. Hansen LK, Salamon P (1990) Neural network ensembles. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 12(10): 993–1001.
24. Hashem S (1993) Optimal linear combinations of neural networks. *PhD dissertation*. School of Industrial Engineering, Purdue University, West Lafayette, IN, December.
25. Deb K (2001) *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, Chichester, UK.
26. Khare V, Yao X, and B. Sendhoff B (2006) Multi-network evolutionary systems and automatic problem decomposition. *Intl. J. General Systems*, 35(3): 259–274.
27. Krogh A, Vedelsby J (1995) Neural network ensembles, cross validation, and active learning. *Neural Information Processing Systems*, 7: 231–238.
28. Krogh A, Sollich P (1997) Statistical mechanics of ensemble learning. *Physics Reviews E*, 55: 811–825.
29. Kwok TY, Yeung DY (1997) Constructive algorithms for structure learning in feedforward neural networks for regression problems. *IEEE Trans. Neural Networks*, 8: 630–645.
30. Kwok TY, Yeung DY (1997) Objective functions for training new hidden units in constructive neural networks. *IEEE Trans. Neural Networks*, 8: 1131–1148.
31. Lehtokangas M (1999) Modeling with constructive backpropagation," *Neural Networks*, 12: 707–716.
32. Lee CY, Yao X (2004) Evolutionary programming using the mutations based on the Lévy probability distribution. *IEEE Trans. Evolutionary Computation*, 8(1): 1–13.
33. Liu Y, Yao X (1999) Ensemble learning via negative correlation," *Neural Networks*, 12: 1399–1404.
34. Liu Y, Yao X, Higuchi T (2000) Evolutionary ensembles with negative correlation learning. *IEEE Trans. Evolutionary Computation*, 4(4): 380–387.
35. MacQueen J (1967) Some methods for classification and analysis of multivariate observation. In: *Proc. 5th Berkely Symp. Mathematical Statistics and Probability*, Berkely, CA, University of California Press, 1: 281–297.
36. Mahfoud SW (1995) Niching methods for genetic algorithms. *PhD Thesis*, Deptartment of General Engineering, University of Illinois, Urbana-Champaign, IL.
37. Monirul Islam M, Yao X, Murase K (2003) A constructive algorithm for training cooperative neural network ensembles. *IEEE Trans. Neural Networks*, 14: 820–834.
38. Mulgrew B, Cowan CFN (1988) *Adaptive Filters and Equalizers*. Kluwer, Boston, MA.

39. Odri SV, Petrovacki DP, Krstonosic GA (1993) Evolutional development of a multilevel neural network. *Neural Networks*, 6(4): 583–595.
40. Opitz DW, Shavlik JW (1996) Generating accurate and diverse members of a neural-network ensemble. *Neural Information Processing Systems*, 8: 535–541.
41. Opitz D, Maclin R (1999) Popular ensemble methods: an empirical study. *J. Artificial Intelligence Research*, 11: 169–198.
42. Perrone MP (1993) Improving regression estimation: averaging methods for variance reduction with extensions to general convex measure optimization. *PhD Dissertation,* Department of Physics, Brown University, Providence, RI, May.
43. Prechelt L (1994) Proben1-A set of neural network benchmark problems and benchmarking rules. *Technical Report 21/94*, Fakultät für Informatik, University of Karlsruhe, Germany, September.
44. Prechelt L (1995) Some notes on neural learning algorithm benchmarking. *Neurocomputing*, 9(3): 343–347.
45. Rissanen J (1978) Modeling by shortest data description. *Automatica*, 14: 465–471.
46. Rumelhart DE, Hinton GE, Williams RJ (1986) Learning internal representations by error propagation. In: Rumelhart DE, McClelland JL (eds.) *Parallel Distributed Processing: Explorations in the Microstructures of Cognition, I.* MIT Press, Cambridge, MA: 318–362.
47. Sharkey AJC (1996) On combining artificial neural nets. *Connection Science*, 8(3/4): 299–313.
48. Schaffer JD, Whitley D, Eshelman LJ (1992) Combinations of genetic algorithms and neural networks: a survey of the state of the art. In: Whitley D, Schaffer JD (eds.) *Proc. Intl. Workshop Combinations Genetic Algorithms Neural Networks (COGANN-92)*, 6 June, Maryland. IEEE Computer Society Press, Los Alamitos, CA: 1–37.
49. Srinivas N, Deb K (1994) Multi-objective function optimization using non-dominated sorting genetic algorithms. *Evolutionary Computation*, 2(3): 221–248.
50. Storn R, Price K (1996) Minimizing the real functions of the ICEC'96 contest by differential evolution. In: Fukuda T, Furuhashi T, Back T, Kitano H, Michalewicz (eds.) *Proc. IEEE Intl. Conf. Evolutionary Computation*, 20–22 May, Nagoya, Japan. IEEE Computer Society Press, Los Alamitos, CA: 842–844.
51. Syswerda G (1991) A study of reproduction in generational and steady state genetic algorithms. In: Rawlins GJE (ed.) *Foundations of Genetic Algorithms.* Morgan Kaufmann, San Mateo, CA: 94–101.
52. Yao X (1991) Evolution of connectionist networks. In: *Proc. Intl. Symp. AI, Reasoning & Creativity*, Griffith University, Queensland, Australia, 49–52.
53. Yao X (1993) An empirical study of genetic operators in genetic algorithms. *Microprocessors and Microprogramming*, 38: 707–714.
54. Yao X (1993) A review of evolutionary artificial neural networks. *Int. J. Intelligent Systems*, 8(4): 539–567.
55. Yao X (1993) Evolutionary artificial neural networks. *Int. J. Neural Systems*, 4(3): 203–222.
56. Yao X (1994) The evolution of connectionist networks. In: Dartnall T. (ed.) *Artificial Intelligence and Creativity.* Kluwer, Dordrecht, The Netherlands: 233–243.
57. Yao X (1995) Evolutionary artificial neural networks. In: Kent A, Williams JG (eds.) *Encyclopedia of Computer Science and Technology 33*, Marcel Dekker, New York, NY: 137–170.

58. Yao X, Shi Y (1995) A preliminary study on designing artificial neural networks using co-evolution. In: Toumodge S, Lee TH, Sundarajan N (eds.) *Proc. IEEE Intl. Conf. Intelligent Control Instrumentation*, 2–8 July, Singapore. IEEE Computer Society Press, Los Alamitos, CA: 149–154.
59. Yao X (1999) Evolving artificial neural networks. *Proc. IEEE*, 87: 1423–1447.
60. Yao X, Liu Y (1996) Ensemble structure of evolutionary artificial neural networks. In: Fukuda T, Furuhashi T, Back T, Kitano H, Michalewicz (eds.) *Proc. 1996 IEEE Intl. Conf. Evolutionary Computation (ICEC96)*, 20–22 May, Nagoya, Japan. IEEE Computer Society Press, Los Alamitos, CA: 659–664.
61. Yao X, Liu Y (1997) A new evolutionary system for evolving artificial neural networks. *IEEE Trans. Neural Networks*, 8(3): 694–713.
62. Yao X, Liu Y (1998) Making use of population information in evolutionary artificial neural networks. *IEEE Trans. Systems, Man, and Cybernetics B*, 28(3): 417–425.
63. Yao X, Liu Y, Darwen P (1996) 'How to make best use of evolutionary learning'. In: Stocker R, Jelinek H, Durnota B (eds.) *Complex Systems: From Local Interactions to Global Phenomena*. IOS Press, Amsterdam, The Netherlands: 229–242.
64. Yao X, Liu Y, Lin G (1999) Evolutionary Programming Made Faster. *IEEE Trans. Evolutionary Computation*, 3(2): 82–102.
65. Yao X, Islam MM (2008) Evolving artificial neural network ensembles. *IEEE Computational Intelligence Magazine*, 3(1) (in press).

Resources

1 Key Books

Fogel DB (1995) *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* IEEE Press, Piscataway, NJ.

Haykin SY (1998) *Neural Networks: A Comprehensive Foundation (2nd ed).* Prentice Hall, Englewood Cliffs, NJ.

Yao X (ed.) (1999) *Evolutionary Computation: Theory and Applications.* World Scientific, Singapore.

Sharkey AJC (ed.) (1999) *Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems.* Springer-Verlag, London, UK.

Kuncheva LI (2004) *Combining Pattern Classifiers Methods and Algorithms.* Wiley, Hoboken, NJ.

2 Key Survey/Review Articles

Kohonen T (1988) An introduction to neural computing. *Neural Networks*, 1(1): 3–16.

Yao X (1999) Evolving artificial neural networks. *Proc. IEEE*, 87(9): 1423–1447.

Brown G, Wyatt J, Harris R, Yao X (2005) Diversity creation methods: a survey and categorization. *J. Information Fusion*, 6: 5–20.

3 Organizations, Societies, Special Interest Groups

IEEE Computational Intelligence Society
http://www.ieee-cis.org/

International Neural Network Society
http://www.inns.org/

European Neural Network Society
http://www.snn.ru.nl/enns/

4 Research Groups

Natural Computation Group at the University of Birmingham, UK
http://www.cs.bham.ac.uk/research/labs/natural_computation/

Machine Learning Research Group (MLRG) at the University of Wisconsin – Madison, USA.
http://pages.cs.wisc.edu/∼shavlik/mlrg/

Neural Networks Research Group at the University of Texas, Austin
http://nn.cs.utexas.edu/

Computer Science and Artificial Intelligence Laboratory, MIT
http://www.csail.mit.edu/index.php

Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA), UK
http://www.cercia.ac.uk/

5 Discussion Groups, Forums

Neural Network Forums
http://www.makhfi.com/cgi-bin/teemz/teemz.cgi

Neural Network Discussion Group
http://itmanagement.webopedia.com/TERM/N/neural_ network.html

6 Key International Conferences and Workshops

Congress on Evolutionary Computation (CEC)

International Conference on Parallel Problem Solving from Nature (PPSN)

Neural Information Processing Systems (NIPS)

International Joint Conference on Neural Networks (IJCNN)

International Conference on Artificial Neural Networks (ICANN)

European Symposium on Artificial Neural Networks (ESANN)

International Conference on Neural Information Processing (ICONIP)

7 (Open Source) Software

Emergent Neural Network Simulation Software
http://neurobot.bio.auth.gr/archives/000116emergent_neural_network_simulation_software_formerly_pdp.php

Forecasting with artificial neural networks
http://www.neural-forecasting.com/

NeuroDimension
http://www.nd.com/

Neural archive at FuNet
http://www.nic.funet.fi/

The PDP++ Software
http://www.cnbc.cmu.edu/Resources/PDP++//PDP++.html

Amygdala for simulating spiking neural networks
http://amygdala.sourceforge.net/

8 Data Bases

Birmingham Repository: Evolutionary Computation Benchmarking Repository (EvoCoBR)
http://www.cs.bham.ac.uk/research/projects/ecb/

UCI repository of machine learning databases
http://www.ics.uci.edu/m̃learn/MLRepository.html

Neural Networks Databases – Benchmarks
http://www.fizyka.umk.pl/neural/node12.html

UCI KDD Archive
http://kdd.ics.uci.edu/

Netlib Repository
http://www.netlib.org/

Statlib
http://lib.stat.cmu.edu/