An Evolutionary Algorithm to Training Neural Networks for a Two-Spiral Problem

Jinn-Moon Yang and Cheng-Yan Kao
Department of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan
e-mail:{moon,cykao}@csie.ntu.edu.tw

Abstract

An evolutionary algorithm is introduced to train neural networks for a two-spiral problem. The proposed approach automatically achieves the balance of the solution quality and convergence speed by integrating multiple mutations, family competition, and adaptive rules. Following the description of implementation details, our approach is applied to a two-spiral problem. Experimental results indicate that the proposed approach is able to stably solve this problem and is very competitive with the comparative evolutionary algorithms.

Keywords: Evolutionary algorithms, family competition, multiple mutations, neural networks, adaptive mutations

1 Introduction

Artificial neural networks (ANNs) have been applied widely in many application domains. ANNs are considered to able to avoid the bias of a designer in shaping system development owing to their flexibility, robustness, and tolerance of noise. Learning the weights of ANNs with fixed architectures can be formulated as a weight training process. This process is to minimize an objective function in order to achieve an optimal set of connection weights of an ANN which can be employed to solve the desired problems.

The standard back propagation learning algorithm [16] and many improved back propagation learning algorithms [6], [10] are widely used to train neural networks. They are the gradient descent techniques which often try to minimize the objective function based on the value of total error between the actual output and the target output of an ANN. This error is used to guide the search of a back propagation approach in the weight space. However, the drawback with a back propagation algorithm do exist due to its gradient descent nature. It may get trapped in local optima of an objective function and is inefficient in searching for a global minimum of an objective function which is vast, multimodal, and non-differentiable [17]. Particularly gradient methods are not directly applicable while the gradient information is not available for some applications. Furthermore, from a theoretical perspective, gradient methods often produce worse recurrent networks than non-gradient methods when an application requires memory retention [3].

An evolutionary algorithm is a non-gradient method and is a very promised approach for training ANNs. It is considered to be able to reduce the ill effect of the back propagation algorithm because an evolutionary algorithm does not require the gradient and differentiable information. Evolutionary algorithms have been successfully applied to train or to evolve ANNs in many application domains [17]. Some pertinent research [14], [20] have demonstrated that the convergence speed of evolutionary algorithms is competitive with the back propagation approach if genetic operators are well designed.

Evolutionary methodologies can be roughly categorized as genetic algorithms [8], evolutionary programming [7], and evolution strategies [1]. Applying genetic algorithms to train neural networks may be unsatisfactory because recombination operators incur several problems, such as competing conventions [17] and the epistasis effect [4]. For better performance, real-coded genetic algorithms [5], [15] have been introduced. In contrast, evolution strategies and evolutionary programming mainly use real-valued representation and focus on self-adaptive Gaussian mutation. This mutation has been widely regarded as a good operator for local searches[7], [1]. Unfortunately, experiments [22] show that self-adaptive Gaussian mutation leaves individuals trapped near local optima for rugged functions.

Because none of these three types of standard evolutionary algorithms is very efficient, many modifications have been proposed to improve solution quality and to

speed up convergence. In particular, a recent trend [9], [12] is to incorporate local search techniques, such as the back propagation approach, into evolutionary algorithms. Such a hybrid approach can combine the merits of an evolutionary algorithm with those of a local search technique. Generally speaking, a hybrid approach usually can make a better tradeoff between computational cost and the global optimality of the solution. However, for existing methods local search techniques and genetic operators often work separately during the search process.

Another technique is to use multiple genetic operators [14]. This approach works by assigning a list of parameters to determine the probability of using each operator. An adaptive mechanism is applied to change these probabilities to reflect the performance of these operators. The main disadvantage of this kind method is that the mechanism for adapting the probabilities may mislead evolutionary algorithms toward local optima.

To further improve the above approaches, in this paper a new method called family competition evolutionary algorithm (FCEA) is proposed for training neural networks. FCEA is a multi-operator approach which combines three mutation operators: decreasing-based Gaussian mutation, self-adaptive Gaussian mutation, and self-adaptive Cauchy mutation. The performance of these mutations heavily depends on the same factor called *step size*. FCEA incorporates the family competition [21] and adaptive rules for controlling *step sizes* to construct the relationship among these mutation operators. The family competition is derived from $(1+\lambda)$-ES [1] and acts as a local search procedure. The self-adaptive mutations adapt its step sizes with a stochastic mechanism based on their performance. In contrast, the decreasing-based mutation decreases the step sizes by applying a fixed rate γ where $\gamma < 1$.

FCEA markedly differs from previous approaches because these mutation operators are sequentially applied with equal probability 1. In order to balance exploration and exploitation, each of these operators is designed to compensate for the disadvantages of the other. To the best of our knowledge, FCEA is the first successful approach to integrate self-adaptive mutations and decreasing-based mutations by using family competition and adaptive rules .

Section 2 describes FCEA in detail and gives motivations and ideas behind various design choices. Sections 3 presents the experimental results of FCEA employed to train neural networks for a two-spiral problem. Conclusions are finally made in Section 4.

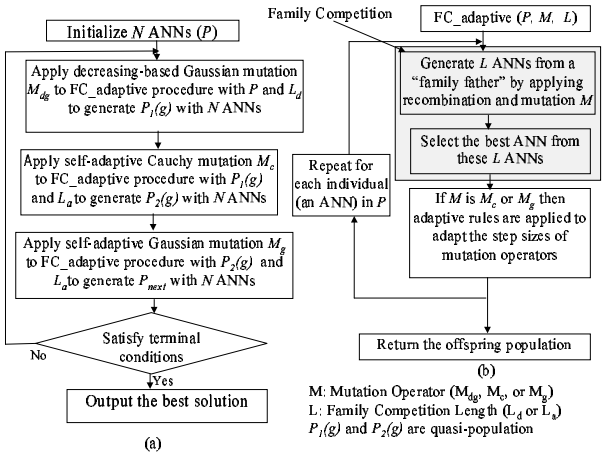


Figure 1: Overview of our algorithm: (a) FCEA (b) FC_adaptive procedure

2 Family Competition Evolutionary Algorithm

In this section, we present the details of the family competition evolutionary algorithm (FCEA) for training neural networks. The basic structure of the FCEA is as follows (Fig. 1): N solutions are randomly generated as the initial population. Then FCEA enters the main evolutionary loop, in which each generation consists of three nearly identical procedures. Each procedure is realized by doing recombinations, mutations, family competition, and selection. These three procedures differ mainly in the mutations used: decreasing Gaussian mutation, self-adaptive Cauchy mutation, and self-adaptive Gaussian mutation. Hence we refer such a procedure as "FC_adaptive" which will be described later in detail. Note that the input of an FC_adaptive procedure is N solutions. Then the output is a new quasi-population with N solutions which will be the input of the next FC_adaptive procedure.

The FC_adaptive procedure (1(b)) employs three parameters, namely, the parent population (P with N solutions), mutation operator (M), and family competition length (L), to generate a new quasi-population. The main procedures of FC_adaptive are the family competition (Fig. 2) and the adaptive rules. During the family competition procedure, each individual in the population sequentially becomes the "family father (I_1)." With a probability p_c, this "family father" and another solution (I_1^1) randomly chosen from the rest of the parent population are used as parents to do a recombination operation. Then the new offspring or the "family father" (if the recombination is not conducted) is operated on by a mutation to generate a offspring (C^{11}). For each family father, such a procedure is repeated L times. Finally L solutions ($C^{11}, \ldots, C^{1L}$) are

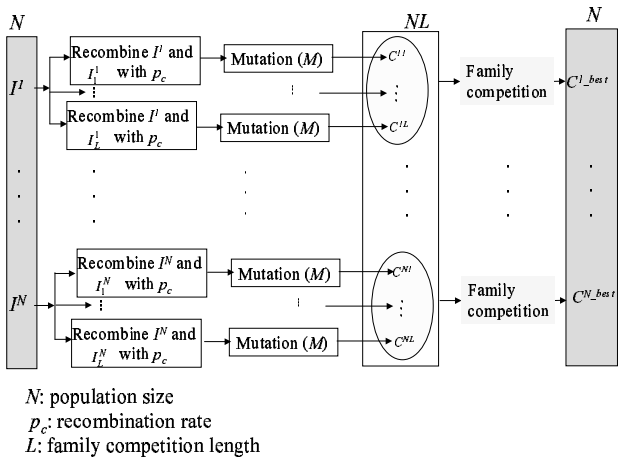


Figure 2: The main steps of the family competition

produced but only the one (C^{1_best}) with the best merit function value survives. Since we create L solutions from the same "family father" and perform a selection, this is a family competition strategy. We thought this was a good way to avoid the premature convergence but also to keep the spirit of local searches, because of these suspicions, and results agree.

After the family competition, there are N parents and N children left. Based on different stages, we employ various ways of obtaining a new quasi-population with N individuals (ANNs). For both Gaussian and Cauchy self-adaptive mutations, in each pair of father and child the individual with a better objective value survives. This is the so called "family selection." On the other hand, "population selection" chooses the best N individuals from all N parents and N children. With a probability P_{ps}, FCEA applies population selection to speed up the convergence when the decreasing-based Gaussian mutation is used. For the probability (1-P_{ps}), family selection is still considered. In order to reduce the ill effects of greediness on this selection, the initial P_{ps} is set to 0.05, but it is changed to 0.5 when the mean step size of self-adaptive Gaussian mutation is larger than that of decreasing-based Gaussian mutation. Note that the population selection is similar to $(\mu+\mu)$-ES in the traditional evolution strategies. Note that we create LN offspring in the procedure FC_adaptive but the size of the new quasi-population remains the same as N.

For all three mutation operators, we assign different parameters to control their performance. Such parameters must be adjusted through the evolutionary process. We modify them first when mutations are applied. Then after the family competition is complete, parameters are adapted again to better reflect the performance of the whole FC_adaptive procedure.

In the rest of this section we explain the chromosome representation and each important component of the FC_adaptive procedure: recombination operators, mutation operations, and rules for adapting step sizes (σ, v, and ψ).

2.1 Chromosome representation and initialization

Regarding chromosome representation, we present each solution of a population as a set of four n-dimensional vectors $(x^i, \sigma^i, v^i, \psi^i)$, where n is the number of weights of an ANN and $\forall i \in \{1, \ldots, N\}$. The vector x is the weights to be optimized; σ, v, and ψ are the step-size vectors of decreasing-based mutations, self-adaptive Gaussian mutation, and self-adaptive Cauchy mutation, respectively. In other words, each solution x is associated with some parameters for step-size control. Herein, the initial value of each entry of x is randomly chosen over $[-0.1, 0.1]$ and the initial values of each entries of the vectors σ, v, and ψ, are set to be 0.4, 0.1, and 0.1, respectively. For easy description of the operators, we use $a = (x^a, \sigma^a, v^a, \psi^a)$ to represent the "family father" and $b = (x^b, \sigma^b, v^b, \psi^b)$ as another parent (only for the recombination operator). The offspring of each operation is represented as $c = (x^c, \sigma^c, v^c, \psi^c)$. We also use the symbol x_j^d to denote the jth component of an individual d, $\forall j \in \{1, \ldots, n\}$.

2.2 Recombination Operators

FCEA implements two recombination operators to generate offspring: modified discrete recombination and intermediate recombination [1]. With probabilities 0.9 and 0.1, at each stage only one of the two operators is chosen. Probabilities are set to obtain good performance according to our experimental experience. Here we would like to mention again that recombination operators are activated with only a probability p_c. The optimizing connection weights (x) and a step size (σ, v, or ψ) are recombined in a recombination operator.

Modified Discrete Recombination: The original discrete recombination [1] generates a child that inherits genes from two parents with equal probability. Here the two parents of the recombination operator are the "family father" and another solution randomly selected. Our experience indicates that FCEA can be more robust if the child inherits genes from the "family father" with a higher probability. Therefore, we modified the operator to be as follows:

$$x_j^c = \begin{cases} x_j^a & \text{with probability 0.8} \\ x_j^b & \text{with probability 0.2.} \end{cases} \quad (1)$$

For a "family father", applying this operator in the family competition is viewed as a local search procedure

because this operator is designed to preserve the relationship between a child and its "family father".

Intermediate Recombination: We define intermediate recombination as:

$$x_j^c = x_j^a + 0.5(x_j^b - x_j^a), \text{ and} \quad (2)$$
$$w_j^c = w_j^a + 0.5(w_j^b - w_j^a), \quad (3)$$

where w is v, σ, or ψ based on the mutation operator applied in the family competition. For example, if self-adaptive Gaussian mutation is used in this FC_adaptive procedure, x in (2) and (3) is v. We follow the work of the evolution strategies community [2] to employ only intermediate recombination on step-size vectors, that is, σ, v, and ψ. To be more precise, x is also recombined when the intermediate recombination is chosen.

2.3 Mutation Operators

Mutations are main operators of the FCEA. After the recombination, a mutation operator is applied to the "family father" or the new offspring generated by a recombination. In FCEA, the mutation is performed independently on each vector element of the selected individual by adding a random value with expectation zero:

$$x_i' = x_i + wD(\cdot), \quad (4)$$

where x_i is the ith connection weight of x, x_i' is the ith variable of x' mutated from x, $D(\cdot)$ is a random variable, and w is the step size. In this paper, $D(\cdot)$ is evaluated as $N(0,1)$ or $C(1)$ if the mutations are, respectively, Gaussian mutation or Cauchy mutation.

Self-Adaptive Gaussian Mutation: We adapted Schwefel's [18] proposal to use self-adaptive Gaussian mutation in training ANNs. The mutation is accomplished by first mutating the step size v_j and then the connection weight x_j:

$$v_j^c = v_j^a \exp[\tau' N(0,1) + \tau N_j(0,1)], \quad (5)$$
$$x_j^c = x_j^a + v_j^c N_j(0,1), \quad (6)$$

where $N(0,1)$ is the standard normal distribution. $N_j(0,1)$ is a new value with distribution $N(0,1)$ that must be regenerated for each index j. For FCEA, we follow [2] in setting τ and τ' as $(\sqrt{2n})^{-1}$ and $(\sqrt{2\sqrt{n}})^{-1}$, respectively.

Self-Adaptive Cauchy Mutation:

We define self-adaptive Cauchy mutation as follows:

$$\psi_j^c = \psi_j^a \exp[\tau' N(0,1) + \tau N_j(0,1)], \quad (7)$$
$$x_j^c = x_j^a + \psi_j^c C_j(t). \quad (8)$$

In our experiments, t is 1. Note that self-adaptive Cauchy mutation is similar to self-adaptive Gaussian mutation except that (6) is replaced by (8). That is, they implement the same step-size control but use different means of updating x.

Decreasing-Based Gaussian Mutations: Our decreasing-based Gaussian mutation uses the step-size vector σ with a fixed decreasing rate $\gamma = 0.97$ as follows:

$$\sigma^c = \gamma\sigma^a \quad (9)$$
$$x_j^c = x_j^a + \sigma^c N_j(0,1) \quad (10)$$

Previous results [22] demonstrated that self-adaptive mutations converge faster than decreasing-based mutations but, for rugged functions, self-adaptive mutations more easily trapped into local optima than decreasing-based mutations.

2.4 Adaptive Rules

The performance of Gaussian and Cauchy mutations is largely influenced by the step sizes. FCEA adjusts the step sizes while mutations are applied (e.g. (5), (7), and (9)). However, such updates insufficiently consider the performance of the whole family. Therefore, after the family competition, some additional rules are implemented:

1. **A-decrease-rule:** Immediately after self-adaptive mutations, if objective values of all offspring are greater than or equal to that of the "family father," we decrease the step-size vectors v (Gaussian) or ψ (Cauchy) of the parent:

   $$w_j^a = 0.97 w_j^a, \quad (11)$$

   where w^a is the step size vector of the parent. In other words, if there is no improvement after self-adaptive mutations, we may propose a more conservative be implemented. That is, smaller step size tends to make better improvement in the next iteration. This follows the 1/5-success rule of (1+λ)-ES [1].

2. **D-increase-rule:** It is difficult, however, to decide the rate γ of decreasing-based mutations. Unlike self-adaptive mutations which adjust step sizes automatically, its step size goes to zero as the number of iterations increases. Therefore, it is essential to employ a rule which can enlarge the step size in some situations. The step size of the decreasing-based mutation should not be too small, when compared to step sizes of self-adaptive mutations. Here, we propose to increase σ if either of the two self-adaptive mutations generates better offspring. To be more precise, after a self-adaptive mutation, if the best child with step size v is better than

its "family father," the step size of the decreasing-based mutation is updated as follows:

$$\sigma_j^c = \max(\sigma_j^c, \beta v_{mean}^c), \tag{12}$$

where v_{mean}^c is the mean value of the vector v; and β is 0.2 in our experiments. Note that this rule is applied in stages of self-adaptive mutations but not of decreasing-based mutations.

3 The two-spiral problem

In the neural network community, learning to tell two spirals apart is a benchmark task which is an extremely hard classification task [13], [6], [19], [10]. The learning goal is to properly classify all the training data (97 points on each spiral as shown in Fig. 3(a)) which lie on two distinct spirals in the x-y plane. These spirals coil three times around the origin and around one another. The data of the two-spiral problem is electronic available from Carnegie-Mellon University connectionist benchmark collection. We follow the suggestion [6] to use 40-20-40 criterion. That is, an output is considered to be a logical 0 if it lies in [0, 0.4], to be a logical 1 if the output lies in [0.6, 1.0], and indeterminate if it lies in [0.4, 0.6]. In testing phase, 20000 points are chosen regularly from the space (i.e. $-10 \leq x \leq 10$ and $-10 \leq y \leq 10$), and the output of the system is defined as in training phase.

Evolution begins by initializing all the connection weights x of each network to random values between -0.1 and 0.1. The initial values of step sizes for decreasing-based Gaussian mutations, self-adaptive Gaussian mutation, and self-adaptive Cauchy mutation are set to 0.4, 0.1, and 0.1, respectively. Table 1 indicates the settings of FCEA parameters, such as the family competition length and the crossover rate (p_c). L_d and σ are the parameters for the decreasing-based mutation; L_a, v and ψ are for self-adaptive mutations. The population size is set to 30.

Lang and Withbrock [13] used a 2-5-5-5-1 network with shortcut connections. Each node is connected to all nodes in all subsequent layers. With one additional bias connection for each node, therefore, there are a total of 138 trainable weights in the network. They employed the standard back propagation approach to train this network, and considered the task to be completed when each of the 194 points in the training set responded to within 0.4 of its target output value.

We follow the work of [13] to solve the two-spiral problem by using the 2-5-5-5-1 network. The fitness function of each individual (an ANN) in FCEA is defined as

$$F = \frac{1}{Tm} \sum_{t=1}^{T} \sum_{k=1}^{m} (Y_k(I_t) - O_k(I_t))^2, \tag{13}$$

where T is the number of training patterns, m is the number of output nodes, $Y_k(I_t)$ and $O_k(I_t)$ are the actual and desired outputs of the output node k for the input pattern I_t. A training input pattern is classified correctly if the tolerance of $|Y_k(I_t) - O_k(I_t)|$ is below 0.4. In this problem, T is 194 and m is 1. The format of a input pattern I_t is (x, y) where x and y denote the axis of the x-y plane.

FCEA executes 10 independent runs for classifying the two-spiral problem, and it successfully classifies all 194 training points in 7 runs. Fig. 4 indicates a convergence curve of FCEA for the two-spiral problem. It correctly classifies 152 and 186 training points at the 5000th and 15000th generation, respectively; it required 25300 generations to learn all 194 training points. Figs. 3(b)-(d) show the two-spiral response patterns of FCEA at the 5000th, 15000th, and 25300th generation, respectively.

For every learning method, an important practical aspect is the number of pattern presentations necessary to achieve the satisfying performance. In the case of a finite training set, a common measure is the number of cycles through all training patterns, also called *epoch*. Table 2 compares our FCEA with previous approaches [13], [6], [10] on the two-spiral problem based on the averaged number of epochs and the ANN architectures. As can be seen, FCEA needs more epochs than constructive algorithm [6], [10], evolving both ANN architectures and connection weights simultaneously, and its convergence speed is also slower than the back propagation approach.

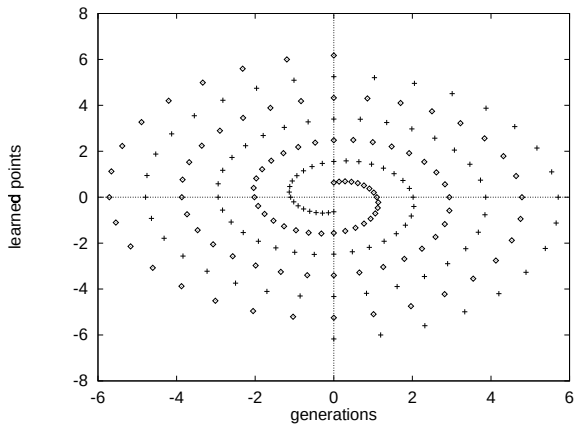
In general, the performance of the back propagation approach is more sensitive to initial weights of an ANN than evolutionary algorithms. The architectures obtained by constructive algorithms seem to be larger than the 2-5-5-5-1 network. To the best of our knowledge, FCEA is the first evolutionary algorithm to stably solve the two-spiral problem by employing the fixed 2-5-5-5-1 network. FCEA is also more robust than genetic programming [11] whose learned points was about 180 on average.

4 Conclusions

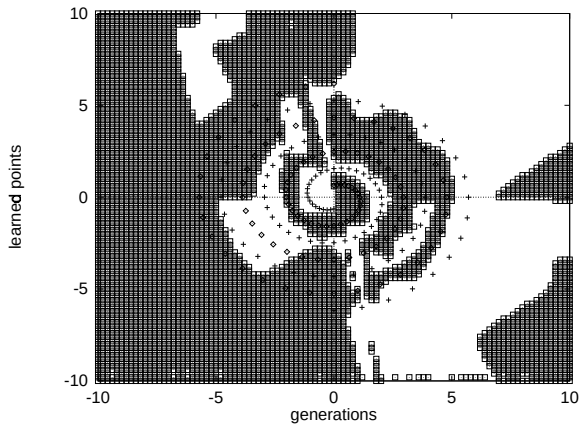
This study presents that FCEA is a stable approach to train neural networks for a two-spiral problem. Our experience suggests that a global optimization method should consist of both global and local search strategies. For our FCEA, the decreasing-based mutation with large initial step sizes is the global search strategy; the self-adaptive mutations with family competition procedure and replacement selection are the local search strategy. Based on the family competition and adaptive rules, these mutation operators can closely cooperate with one another.

Table 1: Parameter Settings and Notations

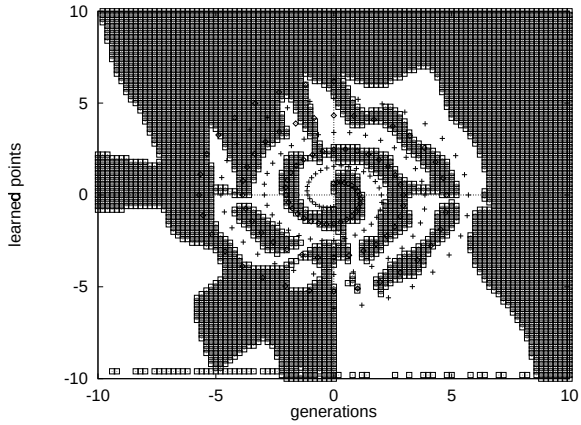
		Step Size		Competition Length		Recombination Rate	
Decreasing-Based Gaussian Mutation	M_{dg}	σ	0.4	L_d	6	p_c	0.2
Self-Adaptive Gaussian Mutation	M_g	v	0.1	L_a	9		
Self-Adaptive Cauchy Mutation	M_c	ψ	0.1	L_a	9		



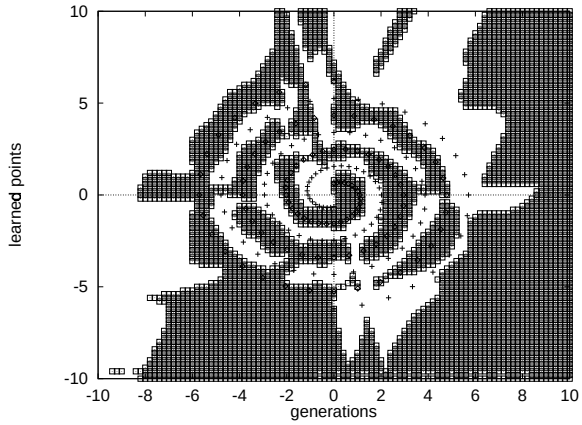
(a) The training data with 194 examples



(b) The classification at the 5000th generation (learned points are 152)



(c) The classification at the 15000th generation (learned points are 186)



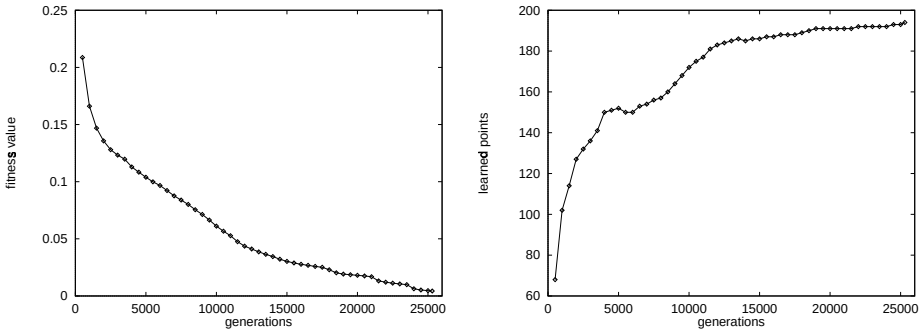
(d) The classification at the 25300th generation (learned points are 194)

Figure 3: The two-spiral problem and the classification solutions created by the ANNs evolved by FCEA at the different numbers of the generations.

Table 2: Comparison FCEA with several previous studies in the two-spiral problem

Method	Number of epochs	Hidden Nodes	Number of Connection weights
Back Propagation [13]	20000	15	138
Cascade-correlation [6]	1700	12-19	N/A
Projection pursuit learning[10]	N/A	11-13	187
FCEA (this paper)	27318 (generations)	15	138

†the values in "number of epochs" and "connection weights" are the average values.
†N/A denotes "not applicable" in the original paper.



(a) Fitness value vs. generations (b) Learned points vs. generations

Figure 4: The convergence of FCEA for the two-spiral problem

Our previous works [23], [24] have demonstrated that FCEA is competitive with the back propagation approach and is more robust than several well-known evolutionary algorithms for several benchmark problems, including the Boolean functions, artificial ant problems, and regular language recognition. According to the experimental results on the two-spiral problem, our FCEA needs more epochs than the better gradient methods (e.g., quickprop [6]) among back propagation approaches. Previous results [12], [17] also described that evolutionary algorithms are less inefficient than back propagation approaches on training complex feedforward neural networks. However, evolutionary algorithms may be a promising learning method when the gradient or error information is not directly applicable, such as the artificial ant problem. For example, our FCEA is very competition with back propagation approaches on training recurrent neural networks for regular language recognition and artificial ant problems [23], [24].

In order to further improve the performance qualities of the FCEA, several modifications and extensions should be investigated in the future. We will extend FCEA to automatically evolve both architectures and connection weights simultaneously because to design a near optimal ANN architecture for some application domains is an important issue. Furthermore, we will investigate FCEA on applications where gradient methods are not directly applicable.

References

[1] T. Bäck. *Evolutionary Algorithms in Theory and Practice.* Oxford University Press, New York, USA, 1996.

[2] T. Bäck and H-P. Schwefel. An overview of evolution algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993.

[3] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. on Neural Networks*, 5(2):157–166, 1994.

[4] Y. Davidor. Epistasis variance: Suitability of a representation to genetic algorithms. *Complex Systems*, 4:368–383, 1990.

[5] L. J. Eshelman and J. D. Schaffer. Real-coded genetic algorithms and interval-schemata. In L. D. Whitley, editor, *Foundation of Genetic Algorithms 2*, pages 187–202. Morgan Kaufmann Publishers, Inc., 1993.

[6] S. E. Fahlamn and C. Lebiere. The cascade-correlation learning architecture. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems II*, pages 524–532. CA:Morgan-Kaufmann, 1990.

[7] D. B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligent.* NJ:IEEE Press, Piscataway, 1995.

[8] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Company, Inc., Reading, MA, USA, 1989.

[9] W. E. Hart. *Adaptive global optimization with local search.* PhD thesis, University of California, San Diego, 1994.

[10] J.-N. Hwang, S.-S. You, S.-R. Lay, and I-C. Jou. The cascade-correlation learning: A projection pursuit learning perspective. *IEEE Trans. on Neural Networks*, 7(2):278–289, 1996.

[11] H. Juillú and J. B. Pollack. Co-evolving intertwined spirals. In *Proc. of the Fifth Annual Conference on Evolutionary Programming*, pages –, 1996.

[12] H. Kitano. Empirical studies on the speed of the convergence of neural network training using genetic algorithms. In *Proc. of the Int. Conf. on Artificial Intelligence*, pages 789–795, 1990.

[13] K. J. Lang and M. J. Witbrock. Learning to tell two spirals apart. In *Proc. of the 1988 Connections Models Summer School*, pages 52–59. Morgan Kaufmann, 1988.

[14] D. J. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. In *Proc. of Eleventh Int. Joint Conf. on Artificial Intelligence*, pages 762–767, 1989.

[15] H. Mühlenbein and D. Schlierkamp-Voosen. Predictive models for the breeder genetic algorithm I. continuous parameters optimization. *Evolutionary Computation*, 1(1):24–49, 1993.

[16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D.E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, pages 318–362. Cambridge, MA: MIT Press, 1986.

[17] 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. of Int. workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 1–37, 1992.

[18] H.-P. Schwefel. *Numerical Optimization of Computer Models*. Chichester: Wiley, 1981.

[19] F. Śmieja. The pandemonium system of reflective agents. *IEEE Trans. on Neural Networks*, 7(1):97–106, 1996.

[20] D. Whitley, T. Starkweather, and C. Bogart. Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel Computing*, 14:347–361, 1990.

[21] J.-M. Yang, Y.-P. Chen, J.-T. Horng, and C.-Y. Kao. Applying family competition to evolution strategies for constrained optimization. In P. J. Angeline, R. G. Reynolds, J. R. McDonnell, and R. Eberhart, editors, *Proc. 6th Annu. Conf. on Evolutionary Programming (Lecture Notes in Computer Science, vol. 1213)*, volume 1213, pages 201–211, 1997.

[22] J.-M. Yang and C.-Y. Kao. Integrating adaptive mutations and family competition into genetic algorithms as function optimizer. *Soft Computing*, 4(2), to appear.

[23] J.-M. Yang, C.-Y. Kao, and J.-T. Horng. A new evolutionary approach to developing neural autonomous agents. In *IEEE Int. Conf. on Robotics and Automation*, pages 1411–1416, 1998.

[24] J.-M. Yang, C.-Y. Kao, and J.-T. Horng. Incorporation of a multi-operator with family competition to training neural networks using a novel evolutionary algorithm. In *Proc. of the Congress of Evolutionary Computation*, 1999.