Combining Competitive and Cooperative Coevolution for Training Cascade Neural Networks

Alexander F. Tulai

Computer Science Dept.
Carleton University
Ottawa, Ont, CANADA, K1S 5B6
alexander.tulai@rogers.com
Tel: (613) 730-2671

Franz Oppacher

Computer Science Dept.
Carleton University
Ottawa, Ont, CANADA, K1S 5B6
foppache@ccs.carleton.ca
Tel: (613) 520-2600/3520

Abstract

Cooperative Coevolution (CC) has been shown to be effective in problems where certain architectural details of the solution are evolved. This is the case of cascade neural networks where the number of hidden units is not pre-established but rather emerges through learning. We take a step towards having coadapted subcomponents emerge rather than being hand designed by showing that competing populations (evolved by GAs with different mutation and crossover probabilities) can be successfully used in selecting the species that are subsequently coevolved in a cooperative model. Our experimental results indicate that retraining is an essential step in the cooperative coevolution model. Previous studies used evolutionary algorithms (EAs) to train connection weights and neuron thresholds in artificial neural networks (ANNs). We show that by also evolving the characteristics of the neurons themselves, the quality of the solution (in terms of number of hidden units) could be significantly improved.

1 INTRODUCTION

EAs have been used in the past to train and/or initialize connection weights, evolve neural network architectures and learning rule adaptation, etc. (Yao, 1999).

This paper shows that in the case of cascade neural networks (CNNs) not only evolutionary strategy (ES) but also genetic algorithms (GAs) could be successfully used for evolving and training the nets. We also show that by evolving the neuron characteristics in addition to the connection weights a more compressed solution is obtained over the case of fix neuron activation function.

The paper is organized as follows. In section 2 we discuss the concept of coevolution. In section 3 we describe the problem under study and the definition of the species used by the EAs. In section 4 we describe the three algorithms that are used for comparison in this study. In section 5 we present and discuss the experimental results. In section 6 we discuss the impact of retraining in cooperative coevolution (CC) while in section 7 we discuss the effect of evolving the neuron characteristics followed by a discussion on algorithm robustness in section 8. Section 9 summarizes the conclusions of this paper.

2 COMPETITIVE AND COOPERATIVE COEVOLUTION

Coevolution is defined as a series of reciprocal evolutionary changes in interacting species acting as agents of selection for each other. Competitive and cooperative coevolution are two important forms of coevolutionary relationships. The nature of the relationship plays an important role in determining various components of the evolutionary model (like problem decomposition, credit assignment, etc.).

Combining competition and cooperation within a coevolution model has been used by cooperative coevolutionary GAs (Potter and De Jong, 1994). In the case of GAs, the competition is usually between individuals and not between populations.

The CC GA model inherits the limitations commonly associated with a GA, like pre-determining, through experimentation, of mutation and crossover rates or the population size. In addition to that, when new species are introduced in the CC model, certain selection criteria or a pool of candidates are needed to ensure the quality of the new species.

In our study, to alleviate these problems and increase the generality of the model, we introduce an extra step during which multiple populations, evolved by GAs with different mutation and crossover rates, participate in a competitive-cooperative coevolutionary process (competing among themselves but cooperating with the previous species) that results in one winning species. The individuals of the populations that underperform during

the competitive phase are re-distributed between the other populations. This process mimics real life situations when employees of a company that goes bankrupt join other successful companies. Consequently, during the competitive phase the competing populations may have slightly different sizes. The size of the population of the winning species will be equal to the sum of all initial populations but it is clear that the mutation and crossover rates are not known apriori as they depend on the GA used by the winning population. The winning species thus joins other previous winners in the cooperative coevolutionary phase of the algorithm. This process is repeated for every hidden unit introduced in the CNN.

The purpose of the competitive phase is twofold. On the one hand it selects a new species and on the other hand it selects the GA with the most appropriate mutation and crossover rates to evolve this new species. The purpose of the cooperative coevolutionary phase is to find those representatives from each species that together provide the best solution to the problem.

We compare the competitive-cooperative coevolutionary (CCC) GA model it with a CC model using evolutionary strategy (ES) and a pool of 8 initial candidate species to select from.

Both algorithms retrain all the species after a new species is introduced in the cooperative. This is different from other similar cooperative models (for example the cascade neural networks architecture can also be seen as a cooperative model) that do not perform retraining.

3 PROBLEM DESCRIPTION AND SPECIES DEFINITION

Proposed first by Alexis Wieland of Mitre Corp. the two-spiral has become a favorite hard problem to solve by training neural networks.

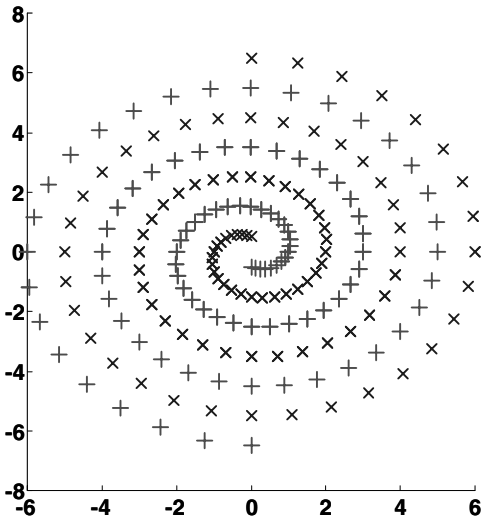

Figure 1 : The training patterns.

The problem consists of two intertwined spirals (see Figure 1) with 97 points on each spiral for a total of 194 points.

If a point that belongs to one spiral is input to the neural network the output of the network should be a positive signal and if the point belongs to the other spiral the output should be negative. When the problem is solved in this form, although it is rarely mentioned, we say that it is solved according to the $50-50$ criterion. Sometimes (Wah and Qian, 2000) classification problems are studied using the $40-20-40$ criterion.

Solutions to the two-spiral problem could either have an evolved architecture, like in the case of the CNNs first proposed by Fahlman (Fahlman and Labiere, 1990a), or they can have a fixed architecture. The best solution to the two-spiral problem, in terms of number of hidden units (HU), had 4 HUs and 19 connection weights and it was arrived at based on a fixed neural network architecture (Wah and Qian, 2000). The previously reported best solution based on an evolved architecture had 9 HUs and 75 weights (Fahlman and Labiere 1990b).

In previous studies (Fahlman and Labiere, 1990a; Potter and De Jong, 2000) the hidden units and the output unit were neurons with an output range of [-1,+1] and with a sigmoidal activation function given by the equation

$$a(x)=\begin{cases}-1, & x<-15\\ \dfrac{2}{1+e^{-x}}-1, & -15\le x\le 15\\ 1, & x>15\end{cases} \qquad (1)$$

We are also using neurons with a [-1,+1] range but we treat the activation function itself as an evolvable function with the equation.

$$a_{\alpha,L}(x)=\begin{cases}-1, & x<-L\\ \tanh(\alpha x), & -L\le x\le L\\ 1, & x>L\end{cases} \qquad (2)$$

Besides the number of hidden units, which is an architectural element, and the connection weights (that also include the bias) we are also evolving the characteristics of each neuron by including the parameters α and L (the gain and the input signal limit) in the individual genome of a species. Defining the species for the cooperative coevolution architecture solution to the two-spiral problem could be successfully done in accordance with the original cascade network training algorithm (Fahlman and Labiere, 1990b). That method consists in, first, evolving together all the connection weights and neuron thresholds (which in fact are also weights on connections to a constant +1.0 input) leading into a new hidden unit and than, second, train all the connection weights leading into the output unit (the two-spiral problem requires only one output). Following this idea (Potter and De Jong, 2000) assign a separate species to the weights leading into a unit whether an output or a hidden unit. This choice of the species has the disadvantage that whenever a new hidden unit is

introduced, a randomly initialized weight needs to be added to all the individuals of the species assigned to the output unit.

In our solution we decided to group all the connection weights needed for a new hidden unit (input, output and bias connections) as well as the neuron characteristics as one species as shown in Figure 2.

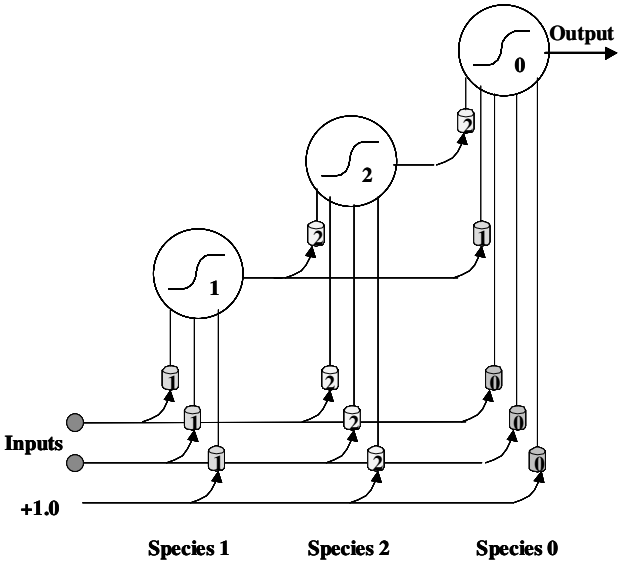

Figure 2 : Cascade net with 1 output and 2 hidden units. All elements of a species have the same number.

Each individual genome is represented by a set of $5+i$, genes where $i = 0,1,\cdots,\max HU$ is the index of the unit introduced, with species 0 representing the output unit and species 1 to maxHU representing the HUs. The first species created represents the output unit and has 5 genes, two for the neuron characteristics and three for the weights on its input connections (one of the connection weights will in fact represent the output unit bias as previously stated). Every time a new hidden unit is introduced, we create a new species with a genome that has one more gene than the previous species. A new hidden unit is always introduced between the last hidden unit and the output unit.

4 ALGORITHM DESCRIPTION

For this study we compare three different learning methods for evolving a cascade correlation neural network that solves the two-spiral problem.

- the GA-based method, that we propose, uses both competitive and cooperative coevolution GA and we will refer to as the CCC-GA method.
- a cooperative coevolution method using the (μ,λ) evolutionary strategy (ES) as described in (Schwefel, 1995) and used by (Potter and De Jong, 2000) for solving the same two-spiral problem. We will refer to this algorithm as the CC-ES method.
- a second order gradient-descent based method as described in (Fahlman and Labiere, 1990a). We will refer to this method as the cascade correlation method.

The cascade networks generated by Fahlman's method are comprised of symmetrical sigmoid units (both hidden and output units) as defined by equation (1). In order to properly compare the CCC-GA and the CC-ES algorithms both will generate cascade networks using neurons with an activation function as defined by equation (2) and will include the parameters α and L among the evolved genes.

All methods try to evolve cascade networks that classify all input patterns, and reach this objective by minimizing the squared error sum (SES) at the output of the cascade neural network over all the patterns in the two-spiral data set,

$$Error = \sum_p (d_p - y_p)^2$$

where p is the training pattern index, d_p is the desired output and y_p is the actual output.

4.1 CCC-GA METHOD

The CCC-GA method has two major phases, a competitive coevolution phase and a cooperative coevolution phase.

The competitive coevolution phase starts with N populations of m individuals, each population being evolved by a GA with overlapping population, and ends with 1 population with $N \cdot m$ individuals at the end of an iterative process that sees the least fit populations being absorbed by the fitter ones. Each GA will use a different mutation and crossover probability described at step 1 of the competitive phase of the algorithm.

Competitive coevolution phase of the algorithm

1. *Initialization*. Create N populations with m individuals in each population. Train each population with a similar GA but use different mutation and crossover probability. In our study N was set to 16, m to 64 and the mutation and crossover probabilities are distinct pairs $(pmut, pxover) \in \{0.125,0.375,0.625,0.875\}^2$.
2. *1-step evolution*. The N populations are each trained on the whole set of input data once.
3. *Ranking*. All $N \cdot m$ individuals are ranked (*0* to $N \cdot m - 1$) based on their fitness $F_i, i = 0,\ldots,N \cdot m - 1$ calculated during training. The population ranking will be done based on the performance of the top M individuals in the ranking individual order with $M \geq m$. The only time when M will be different from m is when $F_m = F_{m-1}$ in which case we must consider all the individuals that have an identical fitness $F_{m-1} = F_m = \cdots = F_{M-1}$.

4. *Credit assignment*. Each of the M individual is assigned a credit Γ_i based on the equation $\Gamma_i = e^{(M-i)/M}, i = 0,\cdots,M-1$. If k individuals have identical fitnesses $F_j, F_{j+1},\cdots,F_{j+k-1}$ with $0 \le j, j+k-1 \le M-1$, they will each be assigned the same credit value

$$\Gamma_l = (\sum_{l=j}^{j+k-1} e^{(M-l)/M})/k$$

Each of the N populations receives a credit equivalent to the sum of the credits received by its own individuals. Please note that if a certain population has no representatives in the top M individual ranking, it receives a credit equal to 0.

5. *Check population elimination criterion*. When a population ranks last D times in succession it is eliminated at step 6, otherwise we continue with step 2. In our study, D is initially set to 10. To avoid possible processing traps, D is decreased by 1 every 10 iterations.

6. *Population elimination*. If the elimination criterion is met, the genomes of the population that is eliminated are one by one distributed between the other $N-1$ populations with a random starting point. After N is decremented by 1 the algorithm continues with step 2 if $N > 1$. The algorithm iterates until only one population with $N \cdot m$ individuals is left. This population, as a separate species, joins the other previous winning species in the cooperative coevolution phase of the algorithm.

Cooperative coevolution phase of the algorithm

7. *Winner integration*. The winning species is cooperatively coevolved with the best representatives from the previous winning species until the SES does not decrease by more than δ from one iteration to another in I consecutive iterations. At any point in time, the number of species cooperating equals the numbers of neurons in the CNN. At the end of the iterative process the genome with the highest fitness is used to grow the size of the cascade correlation neural network with one hidden unit. In our study $\delta = 0.1$ and $I = 10$.

8. *Retraining*. After the new hidden unit is added to the network, an attempt is made to retrain once all the species created up to this point (i.e. all the units of the cascade neural network). During retraining, each species contributes to the network with its best genome except the species that it is retrained. The species are retrained in a random order (for other possibilities see the section on retraining) except the newest species that is always retrained last. The retraining is done using the same stopping criterion as described at point 7. If a species cannot find a better genome through retraining, the previous one is kept.

During step 7 or 8 of the algorithm, if all the points in the training set are correctly classified, the algorithm stops. CCC-GA method uses a GA with overlapping populations (final size 1024 individuals) with a 50% probability of replacement and Tournament selection.

4.2 CC-ES METHOD

To facilitate the comparison of the results of our study with previous results from previous studies, we used the same (μ,λ) evolutionary strategy (ES) as described in (Schwefel, 1995) and used by (Potter and De Jong, 2000) for studying the same two-spiral problem, with the same choice of $\mu = 10$ and $\lambda = 100$.

A very interesting aspect of this evolutionary algorithm is that both the genes under study and the standard deviations used by the mutation operator are part of the genome and consequently are evolved together.

If $x_k^{(0)} \in (-10.0,10.0)^n, k = 1(1)\mu$ are the initial vectors for the connection weights, with n the genome length, we only have to initialize $x_1^{(0)}$ while the algorithm itself will initialize the other $\mu-1$ individuals by addition of $(0,(\sigma_i^{(0)})^2)$ normally distributed vectors. Just as in (Potter and De Jong, 2000) we initialize all the n components of the first standard deviation vector to $20/\sqrt{n}$ while the other $\mu-1$ standard deviation initial vectors are initialized by Schwefel's algorithm.

The n standard deviation components (mutation steps) are updated by multiplying them every generation t with

$$e^{Gauss(0,\sigma_i^{(t)})}$$

the initial values being given by $\sigma_i^{(0)} = 1/\sqrt{n}$. In our study we used the (μ,λ) implementation of the algorithm given by the 'korr2' program. Although the program allows recombination for both components of the parents (connection weights and standard deviations) we turned this option off and used only mutation. The recombination was turned off for a better comparison with (Potter and De Jong, 2000) and because we had also experienced a degradation of the results when it was turned on.

For every new species that is created, with the exception of the first species, the algorithm is executed eight times to generate a pool of eight species, the best genome in each species representing a candidate HUs. The species that produces the genome with the best results in reducing the network residual output error is inserted in the cooperative of species. The number eight has been chosen to match the size of the pool used by the cascade correlation algorithm. The existence of a pool of candidates and the need to pre-determine its size is a disadvantage for the CC-ES method when compared to the CCC-GA method that always runs the competitive coevolution phase only once for all the newly created species. Other cooperative coevolution models that do not use a pool of candidates but rather introduce some criteria of acceptance for the new species have a similar disavantage when compared to the CCC-GA method.

4.3 GRADIENT-DESCENT BASED METHOD

Classical methods for training neural networks have used gradient-descent techniques such as the back-propagation algorithm for a long time.

To speed up the learning process researchers have used a number of schemes like second order methods (some approximate form), conjugate gradient methods and so on. Fahlman's Quickprop algorithm for updating the connection weights also uses a second-order method.

The cascade correlation learning algorithm creates and installs new hidden units in two steps. First it tries to maximize the magnitude of the correlation between the new unit's output V and the residual error signal the algorithm tries to eliminate, which for a certain input data pattern p and a for a certain output unit o is $E_{p,o}$, summed up over all input data patterns and all output units

$$S = \sum_o \left| \sum_p (V_p - \overline{V})(E_{p,o} - \overline{E}_0) \right|$$

where $\overline{V}$ and $\overline{E}_o$ are the values of V_p and $E_{p,o}$ averaged over all data patterns.

The algorithm uses the Quickprop algorithm to update the new hidden unit's input weights while trying to maximize the correlation value S.

In the second phase of the algorithm, the new hidden unit is inserted in the network, its input connection weights are frozen and the input weights for the output unit are re-trained using the same Quickprop algorithm.

When this algorithm is used to solve the two-spiral problem, it starts with a pool of eight candidates whenever it needs to create a new hidden unit.

5 EXPERIMENTAL RESULTS

Fifty runs were performed using the CCC-GA and the CC-ES method because their running time is high. By comparison, the average computer time per run for the cascade correlation method is orders of magnitude smaller and, consequently, we were able to perform 10,000 runs on a comparable timescale.

Table 1**:** Required number of hidden units for the three methods under study

Method	Hidden units				Time[1]
	Min	Max	Mean	Fail	[s]
CCC-GA	7	16	10.54 ± 0.71	0	1620
CC-ES	8	15	9.9 ± 0.57	0	2700
Cascade correlation	10	19	13.4 ± 0.04	7	5

[1] Average running time calculated on a 500 MHz Pentium III PC

Table 1 shows minimum and maximum number of hidden units produced by each method as well as the average number of hidden units produced by each method, along with 99-percent confidence on the mean. If a solution to the two-spiral problem is not found after introducing 25 HUs the run is classified as a failure (non-convergence).

Our results for the cascade correlation based method have been obtained using the 'cascor' program. The results are significantly better than those reported in (Fahlman and Labiere, 1990a) or (Potter and De Jong, 2000) but are similar to those reported by (Hansen and Pedersen, 1994).

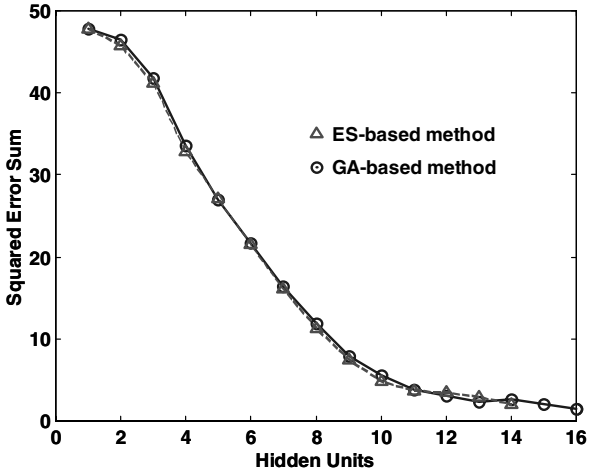

Figure 3: The average decay of the SES as a function of the number of hidden units.

As can be seen in Figure 3, the CCC-GA method and the CC-ES method have very similar behaviour when it comes to decreasing the SES, the two curves being almost identical when the results of 50 runs were averaged. Both CCC-GA and CC-ES methods produce better results than the cascade correlation method, both in terms of average number of hidden units as well as the minimum number of hidden units required for a cascade network solution to the two-spiral problem. However the cascade correlation learning method is a much faster method compared to both coevolutionary methods.

The CC-ES algorithm has produced the best results in terms of average number of hidden units but the CCC-GA produced a cascade network with only 7 hidden units. In fact the neural network with 7 hidden units is the smallest evolved cascade network solution (the number of hidden units is not known apriori) reported in the literature, we are aware of, to the two-spiral problem.

Another important distinction between our results for the CC-ES method and previously reported results (Potter and De Jong, 2000) is that we had no failures in all the runs we performed. In runs where the neuron activation function was given by equation (1) and the training patterns target for classification was set to {-1,+1} we also experienced occasional failures but it was noticed that when the algorithm stopped to converge the output

error was always an integer number. It was determined that when a pattern was misclassified such that the output of the net was minus the desired value (for example –1 instead of +1) the squared error was an integer number. In such cases because of the definition of the sigmoid activation function, any changes to the connection weights that did not result in pushing the output of the net back into the dynamic range of the output unit were not picked up by the algorithm and treated as "better" or "worse" solutions as they should have been. Consequently, the evolution was stuck in a local minimum and the algorithm reduced to a random search. To formally describe the problem, if the neural network is specified by a vector parameter $\mathbf{w} = (w_1, \ldots, w_n)^T \in W^n$, (where the $\mathbf{w}$ is made up of the best genomes from each of the coevolved species and W is the allowed gene range, in our study $[-10,+10]$), the task is to find the optimum vector $\mathbf{w}^*$ corresponding to a net that classifies all patterns in the input set P. When the $50-50$ criterion is used for classification the problem is considered solved when $d_p \cdot y_p(w) > 0, \forall p \in P$.

If $e_p(w) = d_p - y_p(w)$ is the network output error for input pattern $p \in P$ the algorithm achieves the classification goal by minimizing

$$E(w) = \sum_p e_p^2(w)$$

where $y_p(w)$ is the output of the neural network defined by the parameter w and the input pattern p. Let's assume the neural network is comprised of hidden and output units with the activation function given by the equation

$$a\ (x) = \begin{cases} -R, & x < -L \\ b(x), & -L \le x \le L \\ R, & x > L \end{cases}$$

with $\lim_{x \to -\infty} b(x) = -R$ and $\lim_{x \to \infty} b(x) = R$, and for any input pattern there are only two possible outputs used in training $d_p \in \{-R, +R\}$. Let's define an order relation on W^n, $w_1^f < w_2^f \Leftrightarrow E^f(w_1) < E^f(w_2), f \in \{a, b\}$ where the superscript f indicates what activation function is used by the neurons in the network. If $C(m)$ is the set of classified data patterns and $U(m) \neq \varnothing$ is the set of unclassified data patterns after m HUs have been introduced, we have $P = C(m) \cup U(m)$ where P is the set of all input data patterns.

If during training a state is reached where for any pattern $p \in U(m), e_p(w) = 2R$ for any w produced by the algorithm, the EA search space is reduced by all the vectors w_1, w_2 such that $w_1^b < w_2^b$ but $w_1^a = w_2^a$. When this happens *the search becomes purely random* unless the algorithm can find another point w_r such that $e_p(w_r) < 2R, p \in U(m)$ and $E^a(w_r) < E^a(w), \forall w \neq w_r$. Please note this phenomenon could happen because of the limitation on its input signal used by the output neuron activation function $a\ (x)$, but could also happen because of the computational approximation errors in calculating the network output.

Because the two-spiral problem is a classification problem (we shouldn't forget that) the simplest and most effective solution to this problem is to use different values for the desired training values d_p than those used by the activation function as output range limits. For example, in our study $R = 1$ so after we changed the training expected values to $d_p \in \{-1/2, +1/2\}$ the problem has never occurred again and the algorithm has always converged.

6 RETRAINING

A major difference between Fahlman's cascade correlation method and the two cooperative coevolution based methods is the use of retraining. The cascade correlation method, once it has introduced a new hidden unit does not change its input connection weights while both the CCC-GA and CC-ES algorithms use retraining as an important method for further reducing the SES at the network output. For example, when no retraining was used by the CCC-GA method, the performance decreased so severely that only 2 runs were successful in 10 runs performed.

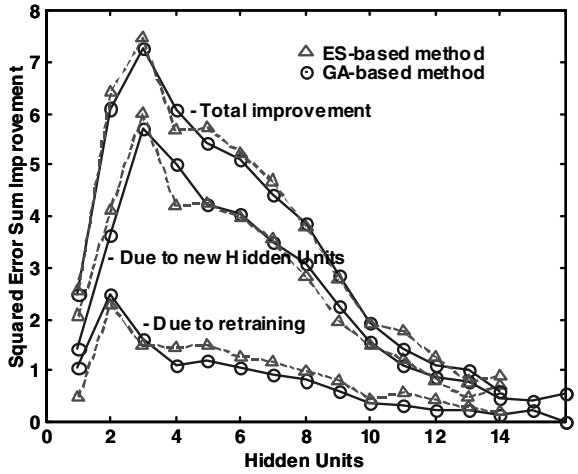

Figure 4: The improvements in squared error sum (SES) after new hidden units, retraining and the sum of the two.

While Figure 4 shows the improvements in SES are very similar for the CCC-GA and CC-ES methods, we could also see that starting with hidden unit 4, on the average, the CC-ES is consistently better in retraining the cascade neural network. This result may be due to the fact that in the case of the CCC-GA method, the GA used to evolve the species that get selected during the competitive coevolution phase, remains fixed during retraining, in the sense that both the mutation and the crossover rate are fixed. The CC-ES uses only mutation, but the mutation vector evolves during retraining just like the individual genes vector does. The search mechanism would benefit from a fully adaptive mutation and crossover probability (White and Oppacher 1994) and the CCC-GA could be modified relatively easily to cope with this limitation. For example, we could do it by randomly dividing the individuals of the species to be retrained in N populations with m individuals in each population,

assigning different mutation and crossover probability to the GA used to train the populations and letting them go again through a competitive phase. However, this would considerably slow down the algorithm in its current form.

Another aspect of retraining we looked at was the order of retraining the species. The results presented in the previous sections were based on retraining all species (including species 0 corresponding to the output unit) in a random order. We however questioned whether retraining done following the order in which the species have been introduced (0 to maxHU-1) or the reverse of that (maxHU-1 to 0) has any impact on the results. The results of the 50 different runs performed with the CCC-GA algorithm are presented in Table 2. In all three cases the last hidden unit introduced (maxHU) is retrained last.

Table 2: Required number of hidden units for CCC-GA when different retraining methods are used

Method	Hidden units			
	Min	Max	Mean	Fail
random	7	16	10.54 ± 0.71	0
$0 \rightarrow \max HU - 1$	8	16	10.42 ± 0.58	0
$\max HU - 1 \rightarrow 0$	8	14	10.46 ± 0.4	0

While the average number of hidden units seems to be similar for the three methods it does appear that the random retraining yields a slightly higher variance of the results. This factor may favour the discovery of the cascade network with 7 hidden units.

7 NEURON CHARACTERISTICS

In our study, the two EA methods have evolved not only the connection weights of the neural network but also the characteristics of the activation functions for all the neurons of the network. To determine the effect of including the characteristics of the neurons as evolvable genes, we have also performed 50 runs where the individual genomes included only the connection weights. In this case all the neurons in the evolved cascade network, the hidden and the output neurons, have a sigmoidal activation function given by equation (1) that does not change during training.

For this aspect of the study we have used only the CCC-GA and CC-ES methods and the results are presented in Table 3.

The results show clearly that including the characteristics of the neurons among the evolved genes not only has increased the generality of the algorithms but it has also significantly improved the results. In the case of the ES-based method the average size of the cascade network has decreased by two hidden units while in the case of the GA-based method the average and the minimum number of hidden units have decreased even more significantly.

Table 3**:** Required number of hidden units with fix or evolvable neuron characteristics.

Method	Hidden units				Neur. char.
	Min	Max	Mean	Fail	
CCC-GA	7	16	10.54 ± 0.71	0	Ev.
CCC-GA	10	17	13.18 ± 0.56	0	Fix
CC-ES	8	15	9.9 ± 0.57	0	Ev.
CC-ES	9	14	11.9 ± 0.45	0	Fix

These results suggest that future studies of neural network evolution should look for a comprehensive solution and also consider evolving the characteristics of the neural activation function.

8 METHOD ROBUSTNESS

The number of adjustable parameters, the sensitivity of the results to user choices, the efficiency of the model in solving different problems, are possible criteria of evaluating the robustness of a computational method.

When comparing the CCC-GA and the CC-ES methods the first thing to differentiate them is the method for choosing the next species to join the cooperative of coevolving species.

While the CC-ES chooses from a pool of 8 different species (why not 4 or 10?), the CCC-GA method doesn't have to make such a choice. In fact, if the CC-ES model doesn't use a pool of species and uses a randomly initialized population, the results (for example the average number of hidden units per evolved cascade network) degrade very significantly. From this point of view it is clear that the CCC-GA method is more robust than the CC-ES method.

When it comes to the evolutionary operators used, ES does not use crossover and the mutation probabilities are independently evolved by the algorithm (so the user has only to initialize an initial vector). While this seems to introduce a clear advantage for the CC-ES method, the CCC-GA compensates for it by evolving the species with the crossover and mutation probabilities best suited for a certain phase of the algorithm. In addition to that, through experimentation we found out that if the ES initial choice is to evolve the mutation probabilities as a group rather than individually (each mutation step evolved independently) the results of the CC-ES will again degrade from the results in Table 1.

Another important parameter influencing the generality and the robustness of the algorithm is the population size. In order to see how the population size influences the results we have performed 50 runs with the CCC-GA with a change in the population size of the species selected for the cooperative phase of the algorithm. Before the species winning the competitive phase is to join the other species

for the cooperative phase of the algorithm we halve the population such that instead of 1024 individuals, each species selected for cooperative coevolution will have only 512 individuals. The results of the two sets of runs are given in Table 4.

Table 4: Required number of hidden units for CCC-GA method with different final populations

Method	Hidden units				Pop. size
	Min	Max	Mean	Fail	
CCC-GA	7	16	10.54 ± 0.71	0	1024
CCC-GA	8	15	10.56 ± 0.62	0	512

The results obtained with species that have only half the population ($N \cdot m/2$) are almost identical to those obtained using species with an entire population ($N \cdot m$) proving that the CCC-GA method is also robust under population variations.

The choice of the GA for the CCC-GA method is not crucial either. A simple GA provides similar results, but it is twice as slow.

Finally, both algorithms need some accuracy of approximation parameters to be set.

While the CCC-GA method needs to be tested on new problems, so far this method has shown robustness and provides a good model for further coevolution studies.

9 CONCLUSIONS

In this study we have introduced the concept of competitive-cooperative coevolution GAs and have shown that, when this method is used for solving the two-spiral problem, the results are very similar to those obtained by using only cooperative coevolution based on an ES-method. The results of both methods, as far as we are aware, are better than any previous cascade networks solutions to the two-spiral problem. It should be noted that previous attempts to use GAs to study the two-spiral problem didn't yield good results. Moreover, in the case of CCC-GA, neither the mutation nor the crossover probabilities are pre-determined (through prior experiments) but are rather selected (from a finite set of possibilities) as a result of the competitive-cooperative coevolution phase of the algorithm.

The cascade neural networks evolved through our methods are complete in the sense that all the characteristics of the nets, number of hidden units, connection weights as well as neuron characteristics are evolved. While evolving the neuron characteristics is natural and easy to do with EA methods it may be more difficult to do in traditional approaches (like the gradient-descent methods). This constitutes a definite advantage of the EA methods for evolving cascade neural networks over other methods.

The CCC-GA algorithm uses the competitive phase to remove the need for a pool of candidates or empirical "goodness" criteria in introducing new species in the cooperative coevolution process and it shows robustness under population and method of retraining variation.

References

S.E.Fahlman and C.Labiere (1990a). The cascade-correlation learning architecture. Technical Report CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania.

S.E.Fahlman and C.Labiere (1990b). The cascade-correlation learning architecture. In D. S. Touretzky (ed.), *Advances in neural Information Processing Systems 2*, 524-532. San Mateo, CA: Morgan Kaufmann.

L.K.Hansen and M.W.Pedersen (1994). Controlled growth of cascade correlation nets. *Proceedings of the International Conference on Artificial Neural Networks*, volume 1, 797—800. Sorrento, Italy.

M.A.Potter and K.A.DeJong (1994). A Cooperative Coevolutionary Approach to Function Optimization. In Y.Davidor and H.P.Schwefel (eds.), *Proceedings of the Third Conference on Parallel Problem Solving from Nature,* 249-257. Berlin, Germany: Springer-Verlag.

M.A.Potter (1992). A genetic cascade-correlation learning algorithm. *Proceedings of COGANN-92 International Workshop on Combinations of Genetic Algorithms and Neural Networks*,122-133. IEEE Computer Society Press.

M.A.Potter and K.A.DeJong (2000). Cooperative coevolution: an architecture for evolving co-adapted subcomponents. *Evolutionary Computations* 8(1):1-20.

H.P.Schwefel (1995). *Evolution and Optimum Seeking*. New York, NY: John Wiley and Sons.

B.W.Wah and M.Qian (2000). Constrained Formulations for Neural Networks Training and Their Applications to Solve the Two-Spiral Problem. *Proceedings of the Fifth International Conference on Computer Science and Informatics*.

T.White and F.Oppacher (1994). Adaptive Crossover Using Automata. In Y.Davidor and H.P.Schwefel (eds.), *Proceedings of the Third Conference on Parallel problem Solving from Nature,* 229-238. Berlin,Germany: Springer-Verlag.

X.Yao (1999). Evolving Artificial Neural Networks. *Proceedings of the IEEE* 87(9):1423-1447.