COOPERATIVE CO-EVOLVING NEURAL NETWORKS FOR ROBOSOCCER SIMULATION

M. Torabi-M, M.-R. Akbarzadeh-T, M. Khademi
Department of Electrical Engineering, Ferdowsi University of Mashhad
torabimm@hotmail.com , Akbarzadeh@ieee.org , khademi@ferdowsi.um.ac.ir

(Received: April 11, 2004– Accepted in Revised Form: Aug. 5, 2005)

Abstract Among various frameworks of intelligence, in general, feed-forward perceptron neural networks (FPNN) is a useful and common method, because of the network's ability to approximate highly nonlinear functions. Similarly, among various paradigms of learning, evolutionary-based algorithms such as genetic algorithms (GA) have gained increasing interest in recent years due to their ability to locate globally optimal solutions in nonlinear, noisy and uncertain problem domains. Here, we propose a cooperative co-evolutionary strategy for finding weights and structure of FPNN simultaneously. The new algorithm allows for separate populations of weights and structures of neural networks to coexist and cooperatively evolve thru two separate genetic algorithms. The proposed algorithm is simulated in RoboSoccer multi-agent environment, and is used for learning the "ball interception" skill of robot soccer players. Also, the convergence properties of the new algorithm are statistically compared with two other approaches as well as standard back propagation (BP) algorithm. Simulation results indicate that the proposed co-evolutionary approach is superior in terms of consistently finding improved solutions.

Keywords Cooperative Co-Evolution, Neural Networks, Weight Optimization, Structure Optimization, Genetic Algorithms, Back Propagation

چکیده شبکه های عصبی از میان ساختارهای متنوع هوشمند, یکی از روشهای بسیار کارا میباشند که توانایی بالایی در تقریب الگوهای آموزشی و روابط غیر خطی دارند. همچنین از میان قالبهای متنوع یادگیری و بهینه سازی, روشهای مبتنی بر اصول تکاملی مانند الگوریتمهای ژنتیک به علت توانایی آنان در بهینه سازی مسائل پیچیده و غیر خطی که عموما همراه با عدم قطعیتها و نویزهای بسیار میباشند, توجه بسیاری را در سالهای اخیر به خود جلب کرده اند. در این مقاله روشی ترکیبی مبتنی بر همکاری هم تکاملی برای بهینه سازی همزمان وزنها و ساختار شبکه های عصبی پیشنهاد میشود, بطوریکه جمعیتهایی مجزا از وزنها و ساختارهای شبکه عصبی بطور همزمان و موازی همکاری نموده و تکامل پیدا میکنند. الگوریتم پیشنهادی بر محیط چند عامله روباتهای فوتبالیست و بمنظور یادگیری مهارت دریافت توپ توسط بازیکنان با وجود نویزهای محیطی قابل توجه مانند وزش باد و رفتار تصادفی بازیکنان شبیه سازی و آزموده شده است. خصوصیات همگرایی این روش از لحاظ آماری با دو روش تکاملی دیگر و همچنین روش پس انتشار خطا(بعنوان معیار) مقایسه شده و برتری آن نشان داده شده است.

1. INTRODUCTION

Artificial neural networks (ANN), among various intelligence paradigms, are considered one of the most potent paradigms for learning and classifying highly nonlinear training patterns. This is while common ways of determining appropriate ANN, and in particular multilayer perceptrons, are still in part ad hoc and in part idealized, i.e. assuming ideal properties of the optimization landscape for

convergence to globally optimal solutions and hence ignoring much of the possible complexities of the application. Hence, a robust and general design methodology which is capable of optimizing both weights and structure of ANN, while the system under study is allowed to contain all of its complexities of both possibilistic and probabilistic nature, would provide a desirable solution to this problem.

Among various algorithms that optimize only weight parameters of an ANN, back propagation (BP) is the most common and popular of supervised algorithms for multilayer perceptrons. BP is a gradient-descent algorithm, which determines the connection weights by back propagating the error in layers of the perceptron such that a given error function is minimized [1,2]. Until today, there have been many successful applications of BP in intelligent systems, control systems, medicine and various other fields [3,4]. Theoretically, BP has been shown to always find the optimum point within limited epochs by properly adjusting training parameters such as learning rate and momentum if the initial weights are set correctly. However, the weakness of BP is that adjustment of these parameters is not a simple task and needs initial information. Additionally, if the error curve is sufficiently complex such that there are many local minimums, BP may never find the global minima.

Unlike BP, evolution-based algorithms such as genetic algorithms (GA) are not easily caught in locally optimal solutions because of their stochastic and parallel exploitation of the optimization landscape [5,6]. Therefore, their application to ANN weight optimization was quite promising. Similar to natural evolution, GA aims to produce progressively better solutions by preferential selection and reproduction of "fitter" individuals (survival of the fittest) and by maintaining diversity through introducing new genetic structures into the population by applying random mutation. One of the advantages of evolutionary algorithms is that they are blind to the problem specifications, i.e. they do not require any problem specific information to build their initial search space. In 1986, Whitley proposed using GA to learn the weights of an ANN [5]. As one of the earlier applications, he demonstrated that GA outperformed the back-propagation algorithm by employing an encoding GA with a relatively high mutation rate [7]. Later, in 1989, Montana and Davis used GA for training a relatively large NN and reported their successful applications [8]. Since then efforts have been made in different ways to improve this technique. Two surveys on the topic of using evolution in optimizing weights of a ANN can be found in [9,10].

While various researches have been directed to determining optimal weights, the problem of optimizing ANN structure remains a challenging problem. In fact, traditional perceptron design is commonly performed by trial and error. Such ad hoc mechanism of finding ANN structure is nontrivial and does not always succeed. Furthermore, there is never a guarantee that the chosen structure is optimal. In recent years several researchers have attended to this problem and proposed several algorithms for determining the structure of multilayer perceptrons. The difficulty is that theoretical estimation of an exact number of hidden neurons is nontrivial, but numerical optimization is available. For example, Ash developed the method of dynamic node creation. A new node is created in hidden layer when the training error rate is above an arbitrarily chosen critical value [11]. In 2000, Peng, et. al. proposed a new hybrid algorithm, which was based on the relationship between the sample approximation error and the number of hidden units. The algorithm also searches the weights [12]. The drawback in above algorithms is that an acceptable sample approximation error is problem specific and is therefore difficult to estimate a priori.

Using evolutionary computing and programming, several algorithms have been introduced in order to determine the optimized structure for neural networks [13,6]. But, leaving many parameters in a bundle for an evolutionary algorithm to organize and optimize presents several problems. Even though, these algorithms are evolutionary, they may still fall prey into premature convergence and problem of competing conventions. The problem of competing conventions is particularly prevalent here because of large number of interdependent parameters when optimizing structure and weights of an ANN using standard evolutionary technique.

Co-evolutionary optimization, in comparison, attempts to divide a difficult problem into simpler sub-problems while remaining a population based

evolutionary search engine. In this fashion, it attempts to avoid the problem of competing conventions. Hence as will be shown in this paper, it demonstrates a great potential to solve complex problems. Co-evolution refers to the simultaneous evolution of two or more species with strongly coupled fitness. In terms of solving engineering problems, if a problem's complex parameter space can be separated, co-evolution allows for searching the reduced parameter spaces in parallel. Such parallel exploration and exploitation of the parameter space can be expected to reduce the problem of competing conventions, and hence yield increased performance and favor the discovery of optimal solutions. In cooperative co-evolutionary algorithms, a number of independently evolving species cooperate to find fitter coupled solutions. The fitness of an individual depends on its ability to collaborate with individuals from other species [14, 15, 16].

Co-evolutionary algorithms have attracted many researchers in recent years. Potter first proposed cooperative co-evolution in 1994 as a general function optimization approach [17]. Later in 1995, Potter presented a co-evolutionary architecture for solving decomposable problems and applied it to the evolution of weights of artificial neural networks [18]. The co-evolutionary approach utilized a divide-and-conquer technique in which species representing simpler subtasks are evolved in separate instances of a genetic algorithm executing in parallel. Collaborations among the species are formed representing complete solutions. In 2001, Reyes and Sipper proposed their co-evolutionary algorithm 'Fuzzy Co-Co' for simultaneously determining the membership functions and rule sets in a fuzzy system [15]. In 2000, Potter and Dejong developed a model in which a number of populations explored different decompositions of the problem. They concluded that their approach adequately addresses issues like problem decomposition and interdependencies between subcomponents [20]. Paredis [14] applied cooperative co-evolution to problems which involved simultaneous search for values and order of pieces of a solution. In the current literature, several other co-evolutionary algorithms can be mentioned for simultaneously finding the unknown quantities in a problem [21, 22]. A general characteristic of above co-co algorithms is that they tend to decompose a problem into many smaller sub-problems, hence replacing the complexity of the original problem by the complexity of handling the interactions among many simpler sub-problems.

In 1997 and later in 2003, Akbarzadeh, et. al. [19, 20] proposed a co-evolutionary mechanism for optimizing fuzzy systems in which two subpopulations (parameters of membership functions and rule structures) were co-evolved by two different evolutionary paradigms, GA and GP. There, GA was advocated as a good paradigm for optimizing numeric strings, while GP was a good paradigm for optimizing rule-based structures. This strategy provided for a reasonable balance between problem decomposition and complexity of interaction. In this paper, we continue this strategy of simple cooperation by proposing a general cooperative co-evolutionary strategy for finding weights and structure of FPNN simultaneously as shown in Figure 1. The new algorithm allows for separate populations of weights and structures of neural networks to coexist and to cooperatively evolve in parallel by two genetic algorithms [21]. The first GA is real-valued in order to optimize weights of FPNN, while the second GA is binary valued in order to optimize discrete number of neurons in each of layers. The proposed algorithm can be expected to help avoid premature convergence and competing conventions. The proposed algorithm is simulated in RoboSoccer multi-agent system environment, and is used for learning the "ball interception" skill of robot soccer players. Statistical analysis of simulation results and comparisons with two other algorithms indicate that the proposed co-evolutionary approach is superior in terms of consistently finding improved solutions.

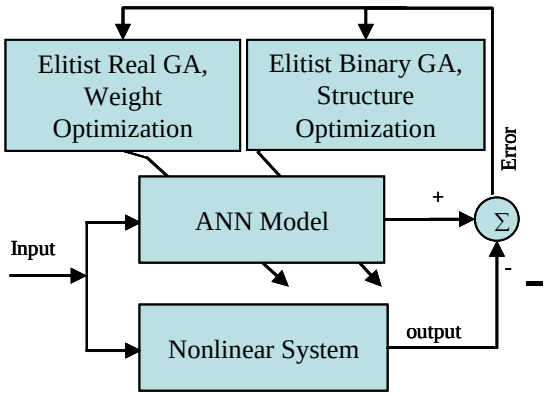

Figure 1. Co. Co. GA for optimization of Neural Networks models

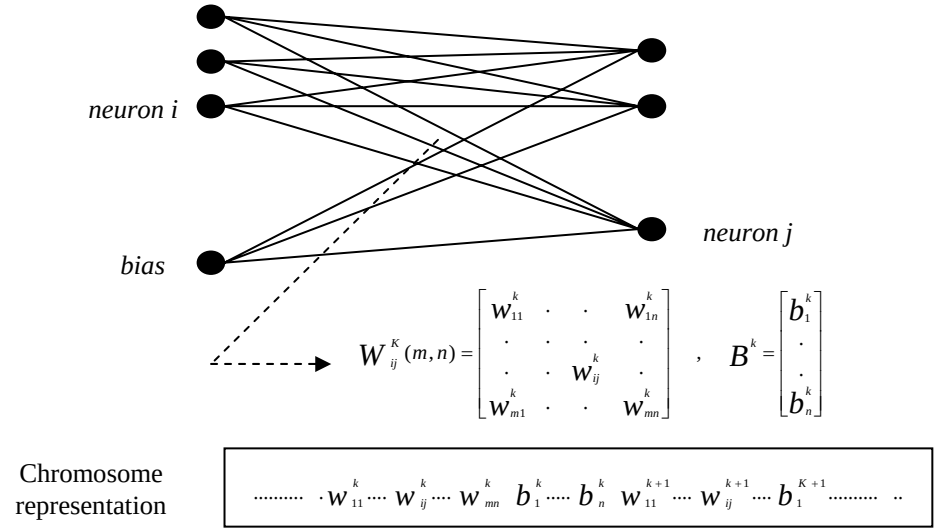

Figure 2. The chromosome representation in first GA algorithm(species 1)

This paper is organized as follows. Section II discusses the evolutionary process in species 1, which searches for optimized weights. Section III discusses the evolutionary process in species 2, which searches for optimized structure of perceptron. Section IV illustrates the co-evolutionary strategies executed for calculating

bonding matrix of fitness. In Section V, the algorithm is simulated and used for learning the "ball interception" skill in soccer playing robots. Statistical analysis of results is then provided in section VI.

2. FIRST SPECIES: EVOLUTIONARY SEARCH FOR DETERMINING THE WEIGHTS

Conventionally, binary-valued GA is applied to parameter optimization for problems even if the parameter space is real itself. However, such approach lacks adequate precision when it is used for determining weights of ANNs [23], while increasing the number of significant bits causes delayed convergence. Therefore, a real-valued genetic algorithm is proposed here for optimizing the first species (weights) in this co-evolutionary algorithm. In the following, chromosome representation and standard GA operators are discussed. Since fitness is a function of both weight parameters and network structure, fitness evaluation is discussed later in section IV.

2.1. Chromosome representation

The connection weights among neurons in each layer of a FPNN are considered as parameters of the real-valued GA. The structure of the ANN, i.e. the number of hidden layers and the number of neurons in each layer, is assumed to be either known a priori or determined by some other mechanism as discussed in next section. For the purposes of simulations in this paper, the transfer function of neurons in the hidden layer is sigmoidal and the output neuron is linear. However, the methodologies are general and can be utilized with an array of possible alternatives. Each chromosome consists of elements of the

weight matrices and biases. Figure 2 shows how one chromosome is represented. In this figure each weight is expressed as W_{ij}^{k} in which i and j indicate the connected neurons and k indicates the layer number. Generally speaking, having a multilayer perceptron with n hidden layers, one input and output layer, the length L of each individual in the search space is calculated as,

$$L = \sum_{k=0}^{n-1} (n_k + 1) n_{k+1} \quad (1)$$

where n_k is the number of nodes at k^{th} layer, $k = 0,...,n$ with $k=0$ being input layer. The initial population is generated randomly between parametric bounds of connection weights and biases.

2.2. Genetic operators

Reproduction, crossover and mutation are the three main operators in a real-valued genetic search as defined below.

Reproduction: Individuals duplicate themselves into an intermediate generation. Thus, higher fit individuals stand a better chance of reproducing while lower fit individuals are more likely to disappear. For reproduction, here, roulette wheel is used in which selection probability is assigned to each individual according to its fitness based on the following probability,

$$P[i\text{th individual being selected}] = \frac{f(i)}{\sum_{k=1}^{pop-size} f(k)} \quad (2)$$

Where $f(i)$ is fitness of i^{th} individual and *pop-size* is the population size. Roulette wheel reproduction has a disadvantage, i.e. highly fit individuals may not be selected for reproduction and may be lost in this process. To improve this weakness, the roulette wheel algorithm is combined with "elitism," such that a certain percentage $\%P$, usually between 3% to 5%, of the best individuals in each generation are allowed to be directly transferred to its future generation, bypassing selection and other genetic operators.

Crossover: Two or more individuals combine to generate new individuals. Crossover enables the evolutionary process to exploit promising regions of the search space, and converge. In this paper, a one-point crossover is used, in which one section of the parent's chromosomes are exchanged at randomly selected points.

Mutation: Mutation is introduced to prevent premature convergence to locally optimal solutions by exploring new points in the search space. By mutation, each real-coded allele in a chromosome may be modified by a random value within its predefined parametric bounds.

3. SECOND SPECIES: EVOLUTIONARY SEARCH FOR DETERMINING THE STRUCTURE

While the Real GA in Section II seems to be an adequate framework for optimizing weights, it does not provide an adequate mechanism for optimizing the structure. Since the number of rules is discrete, binary GA is advocated as a more efficient evolutionary approach for species 2 (FPNN structure). This algorithm is general and can be used for all other structural optimization purposes of neural networks. It also can be combined with any weight optimization method such as BP in order to optimize both weights and structure of an ANN [25]. Chromosome representation and genetic operators in species 2 of the co-evolutionary algorithm are discussed below. Similar to previous section, fitness evaluation is discussed separately in section IV.

3.1. Chromosome representation

An elitist binary GA determines the number of neurons in each of the (two, in this simulation) hidden layers, while also allowing for possible representation of solutions with only one hidden layer. Considering the fact that perceptrons can learn nonlinear continuous training patterns with only one hidden layer [1], two hidden layers should provide for a sufficiently diverse optimization landscape. Hereafter, by the term "layer" we simply mean the number of hidden layers. Obviously, the network has one input and one output layer, and number of neurons in these two layers is determined by the specific learning problem. Figure 3 shows the process of making search population. The maximum number of

neurons M_i in each layer i is determined by considering the features of the problem such as the number of training data or the desired minimum error. The transfer function of each neuron is defined a priori.

3.2. Genetic operators

Standard genetic operators, crossover, mutation and reproduction, are used here, as in real GA algorithm. These processes were also briefly discussed in section II.

4. COOPERATION IN EVOLUTION

Even though the algorithms in section II and III take advantage of GA search, either of search routines only provide partial solutions for ANN and need to cooperate with the other search routine for their fitness evaluation and composing complete solutions. Here, we focus on how the two algorithms cooperate in forming a whole individual and in evaluating their own individual fitness as shown in Figure 4.

4.1. Fitness evaluation

Cooperation among the individuals of the two genetic algorithms is necessary in order to determine each individual's fitness level from their joint fitness functions. In the proposed cooperative genetic algorithm, i^{th} individual in one population (of potential weight solutions) bonds with j^{th} individual of the other population (of potential ANN structures) to compute their joint fitness, f_{ij}. The resulting joint fitness comprises elements of a bonding fitness matrix F, with number of rows and columns equal to the number of individuals in the two populations. As shown in Figure 4, fitness of each *weight* individual is determined by the average of its bonding with individuals of the other subpopulation, similarly the mean fitness value in each row represents fitness of each *structure* individual. The population size of species 1 and 2, in this example, are set at 30 and 100 respectively. Since there are different numbers of neurons in different structures, the length of the *weight* individuals is also variable.

$$F=\begin{array}{c} \begin{matrix} 1 & 2 & . & . & . & . & 100 \end{matrix} \\ \begin{bmatrix} f_{1,1} & . & . & . & . & f_{1,100} \\ . & . & . & . & . & . \\ . & . & f_{i,j} & . & . & . \\ f_{30,1} & . & . & . & . & f_{30,100} \end{bmatrix} \end{array} \begin{matrix} 1 \\ 2 \\ . \\ . \\ . \\ 30 \end{matrix}$$

$$\Rightarrow \text{fitness } i^{th} \text{ weight} = \frac{\sum_{j=1}^{30} f_{i,j}}{30}$$

$$\Rightarrow \text{fitness } j^{th} \text{ structure} = \frac{\sum_{i=1}^{100} f_{i,j}}{100}$$

Figure 4. Bonding Matrix in Co.Co. GA

To solve this problem, a maximum length is determined for the weight population. For instance, assuming $M_i = 10$, the maximum length L will be 161 from Equation 1 (In this example we have assumed 3 inputs at the input later, 1 output at the output later, and two hidden layers with variable length). Individuals, usually, do not fully utilize this allowed space. For example, the weight individuals use 73 parameters of the maximum length when coevolving with the structure (8,4) and with the structure (4,0) use 21 parameters of the maximum length. Figure 5 shows the flowchart of the hybrid GA/BP algorithm.

5. LEARNING SOCCER PLAYING SKILLS IN A MULTI AGENT ENVIRONMENT

The test bench in this research is a robotic soccer simulation environment, provided by "Soccer Server" simulator version 7.9. RoboSoccer simulation is a popular multi-agent environment for testing various paradigms of intelligence and particularly addressing various issues of multi-agent systems. The simulator is based on a client-server model in which the server models the real world and reports the state of the world while the clients control the individual agents. The simulator describes the current state of the world to the clients (agents). The clients periodically send their commands to the simulator indicating how the agents should play.

The goal of using the proposed algorithms here in RoboSoccer environment is learning "ball interception skill". The "ball interception skill" is a

basic individual behavior which all players should possess. This is while receiving moving balls in soccer server simulator is not trivial because of different noise sources in
the environment such as wind velocity, uncertainties regarding the ball and noise in behavior of the players affecting the ball and changing its direction. The RoboSoccer simulator therefore adds a certain amount of noise to the movement of each mobile component, i.e. both the ball and the soccer players, the amount of which is determined by different parameters in soccer server

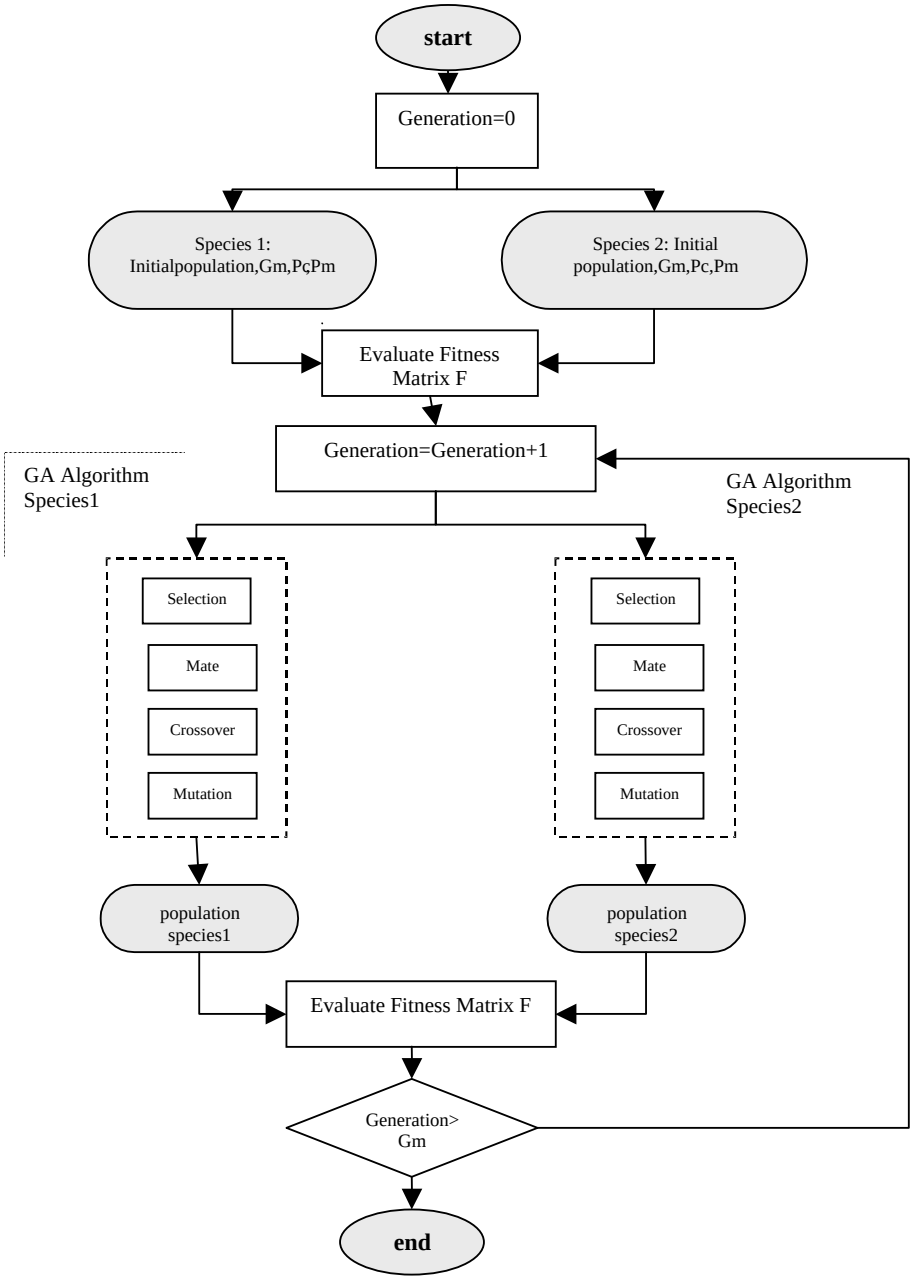

Figure 5. Shows the flowchart of Co.Co. GA algorithm

simulator. Here, the SoccerServer's default values as used in the RoboCup competition are used, as follows:

Player-rand (the noise coefficient which affects

the player's motions) = 0.1
Ball-rand (the noise coefficient which effects the ball motion) = 0.05
Kickable-margin (the area in which the ball is caught by player) = 1.
With the above randomness, environmental noise is so extreme that if a player moves directly towards the ball in order to intercept it, he will be able to intercept it only 30% of the times [26,27].
Hence, in order to catch the ball successfully, players must learn to predict the path of the ball using proper decision parameters and turn to the *predicted* direction, as shown in Figure 6.
In order to train the co-evolutionary FPNN as proposed in this paper, Stone's algorithm [27] is used to create a training set of successful instances of ball interception. In Stone's algorithm, ANN has three inputs as follows,

1. The ball's distance at time $k = d_B(k)$
2. The ball's distance at time *k-1* $= d_B(k-1)$
3. The ball's relative angle at time $k = \theta_{BP}(k)$

The parameter k is the time the agent should decide, turn and move towards the ball. And the ANN output $\hat{\theta}_T(k)$ is the angle by which the robot should turn at time k in order to intercept the ball successfully. In this algorithm there is a shooter agent who shoots the ball, a defender agent who tries to catch the ball and a trainer who saves the results and sends commands. Using Stone's proposed algorithm and running the algorithm_in virtual situation, 1000 successful interceptions are collected for training purposes. Each successful interception try consists of 3 inputs for neural network (as discussed above) and the suitable angle by which the agent has turned to intercept the ball successfully as the network output. The training set is gathered by trainer and saved in a file in order to be used for training our network. This number of training data was quite sufficient for proper training. As will be shown in the simulation results, using too many data points can lead to over-training and impede performance.
Simulations in the following section will test learning performance under various training patterns.

6. SIMULATION AND RESULTS

Once training is completed, in order to test the proposed NN training algorithm, the success of the trained agents in intercepting the ball are evaluated for 3000 trials in the RoboSoccer virtual environment. Here the receiver predicts its proper turn angle using the output of ANN. Table 1 shows the results of the cooperative GA simulation. The second row displays the proposed structure and the third row represents the success rate. In this simulation different sizes of training sets are used and the results are presented in Table 1. As this table illustrates, the highest success rate is obtained by 700 training patterns. In cases of using less than 300 training patterns, results are not as good because there are still many unseen situations for the network whereas employing more than 700 training patterns reduces the success rate because of "overtraining." Figure 7 shows the average and maximum fitness functions of populations in the proposed algorithm.
Table 2 illustrates a comparison of the proposed algorithm with three other algorithms, by enumerating the percentage of success of their best solution. Column 2 shows the results of Hybrid GA/BP simulation that searches for optimal structures using evolution and determines the weights using BP algorithm [24]. Column 3 shows the result of an elitist real GA which optimizes ANN weights using evolution considering a fixed structure [25]. Column 4 displays the result of training the network with BP as reported by Stone using a static ANN with 4 neurons in hidden layer [27]. Column 5 displays the behavior of receiving agent when he has not been trained and simply

Table 1. Percentage of Successful Interceptions (with 700 training data pairs & different solutions)

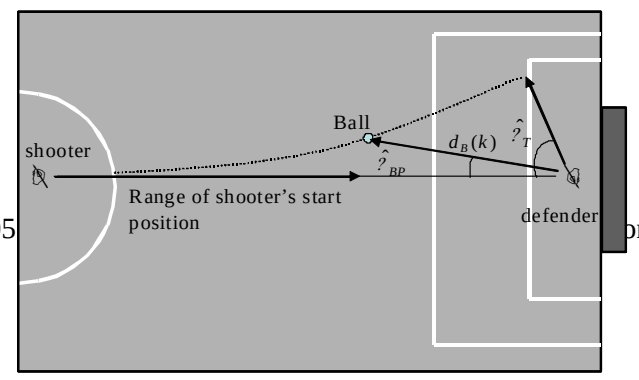

Figure 6. A robot soccer field with a shooter and a defender

GA parameter: Pop1-Size=40 , Num_Generation1=30 , Pc1=0.8 , Pm1=0.08 Pop2-Size=100 , Num_Generation2=250 , Pc2=0.4 , Pm2=0.08		
Input Data	Proposed Structure layer1 layer2	Percentage of Success(%)
100	5 3	82.5
300	6 3	85.5
500	6 4	87
700	6 4	89.5
800	8 3	86.4

Table 2. Percentage of Successful Interceptions (with 700 training data pairs) during testing

	Proposed Cooperative GA	Hybrid GA/BP	Elitist Real GA	Backpropagation	Without training
Percentage of Success(%)	89.5	88	86.2	86	32

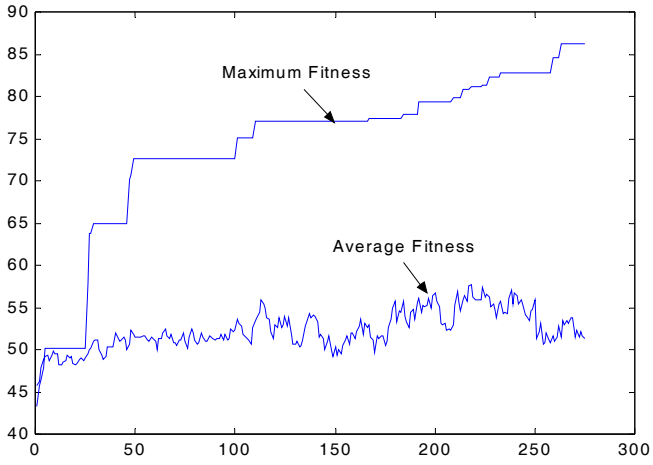

Figure 7. Co.Co.GA Fitness Performance
Pop-Size=100, Generation=250

moves towards the ball. In order to compare consistency, computational intensity and final performance of the three evolutionary algorithms during training, simulations are repeated ten times. Table 3 displays the final fitness of best individuals in the ten trials of the algorithms (to show final

performance), standard deviation of the best individuals (to show consistency), and number of floating point operations (FLOPS) performed during the ten algorithms (to show their computational intensiveness). FLOPS are a reasonable measure of comparing computational intensiveness as it is independent of the processor, hence it will make future comparisons easier. As indicated in Table 3, Cooperative GA finds better optimal solutions more consistently among the three algorithms, i.e. it determines the best solution (comprising of both structure and weights) more consistently. Because of random nature of evolution, it is guaranteed that globally optimal solutions will ultimately be found, given enough generations and sufficiently inclusive search space. However, as Table 3 also indicates, the only disadvantage with the proposed Cooperative GA is that the algorithm is computationally intensive, as simultaneous execution of two GAs is considerably time consuming. Figure 8 shows the performance of Cooperative GA in comparison with three other algorithms. In this figure the MSE error is shown with respect to generation. Cooperative GA has reached the least MSE.

Table 3. Fitness Comparison During Training (obtained by 10 independent runs)

	Best	Worst	Average	Standard Deviation	FLOPS
Co.Co. GA	88	79	84.05	2.99	1790025750
Hybrid GA/BP	68	48	53.8	3.1	150262895
Elitist Real GA	52.4	40	50.3	3.64	13152123

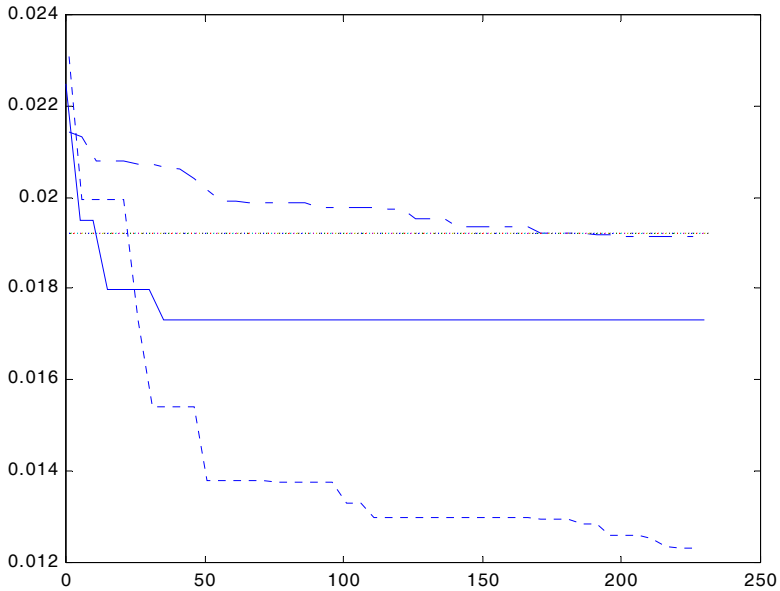

Figure 8. Performance comparison versus generation

7. CONCLUSION

In this paper, a novel cooperative GA is proposed for optimizing ANNs, both parametrically and structurally. For testing purposes, the algorithm is applied to learning ball interception skill in a simulated multi-agent RoboSoccer environment. The algorithm is compared with three other algorithms used for optimizing the ANN and also with an untrained agent. Simulation results indicate that there is a trade off between finding globally optimal solutions and computational intensity. The ten simulation runs that are performed showed that the proposed Cooperative GA consistently finds better solutions (with a lower standard deviation), but is significantly more time consuming. If a problem, such as the one solved in this paper, can be optimized offline, and small amount of improvement in performance and accuracy is significant, then Cooperative GA is a method of choice.

8. NOMENCLATURE

f(i)	Fitness of i^{th} individual
P(i)	Probability selection of i^{th} individual
M_i	Maximum number of neurons in i^{th} layer
f_{ij}	Joint fitness of individuals i and j
Pc1	crossover probability of species1
Pc2	crossover probability of species2
Pm1	mutation probability of species1
Pm2	mutation probability of species2
Pop1-Size	population size of species1
Pop2-Size	population size of species2

9. KNOWLEDGMENT

The cooperation of RoboCup simulation team of Ferdowsi University of Mashhad, and in particular Mr. Moghaadszadeh and Mr. Bagheri, is gratefully acknowledged.

10. REFERENCES

1. Simon Haykin, *Neural Network: A Comprehensive Foundation*, Prentice Hall, New Jersey, 1999.
2. C. Lu & B. Shi and L. Chen, "Hybrid BP-GA for multilayer feedforward neural networks," *Proceedings of the 7th IEEE International Conference on Electronics, Circuits and Systems*, 2000. ICECS 2000, Volume: 2 , 17-20 Dec. 2000, Page(s): 958 -961 vol.2
3. Y. Liu & X. Yao, "Evolving artificial neural networks for medical applications," *Proceedings of first Korea-Australia Joint Workshop on Evolutionary Computation*, Korea, pp. 1-16, 27-29 September, 1995.
4. L. Davis, "Mapping neural networks into classifier systems", *Proceeding of the 3rd Int. Conf. On Genetic Algorithms* (ICGA,89), George Mason University, pp. 375-378,1989.
5. D. Whiteley, "Applying genetic algorithms to neural networks learning," *Proceeding of 7th Conference of the society of Artificial Intelligence and Simulation of Behavior*, Sussex, England: Pitman Publishing, pp. 137-144, 1986.
6. L.C.Jang, *Evolutionary Computing in NN. Design*, pre-published copy, 1997.
7. D. Whiteley, T. Starkweather & C. Bogart, "Genetic algorithms and neural networks: optimizing connections and connectivity," Parallel Computing, Vol. 4, pp. 374-361, 1990.
8. D. Montana & L. Davis, "Training feedforward neural network using genetic algorithms," *Proceeding of 11th of Int. Joint Conf. On Artificial Intelligence*, San Mateo, CA, pp. 762-767, 1989.
9. X. Yao, "A review of evolutionary artificial neural networks," International Journal of Intelligent Systems, Vol. 8, pp. 539-567, 1990.
10. J.D. Schaffer & D. Whiteley and L.j. Eshelman, "Combinations of genetic algorithms and neural networks: A survey of the state of the art," International Workshop on Combination of Genetic algorithms and neural networks, Baltimore, Maryland, June 6, 1992.
11. T. Ash, "Dynamic node creation in backpropagation networks," *Proceeding of Int. Conf. On Neural Networks*, San Diego, 1989.
12. K. Peng & S. Ge and C. Wen, "An algorithm to

determine neural network hidden layer size and weight coefficients," *Proceeding of the 15th IEEE Int. Symposium on Intelligent Control*, July 17-19, 2000.
13. Z. Liu & M. Sugisaka, "A genetic algorithm approach used to generate the neural network structures," *Proceeding of the Int. Conf. On Intelligent Robotics and Systems* (RSJ), IEEE, pp.763-768, 1999.
14. J. Paredis, "Coevolutionaty computation", Artificial Life, Vol. 2, pp. 355-375, 1995.
15. C. Andres Pena-Reyes and M. Sipper, "Fuzzy CoCo: A cooperative coevolutionary approach to fuzzy modeling," IEEE Transaction on Fuzzy Systems, Vol. 9, No. 5, pp. 727-737, 2001.
16. Jonghyeok jeong and Se-Young Oh, "Automatic rule generation for fuzzy logic controllers using rule-level co-evolution of sub populations," In *Proceedings of the Congress on Evolutionary Computation*, Washington, DC, July 1999
17. Mitchell A. Potter and Kenneth A. De Jong, "A cooperative coevolutionary approach to function optimization," In *Proceedings of the Third Conference on Parallel Problem Solving from Nature*, pp. 249-257. Springer-Verlag, 1994.
18. Kenneth A. De Jong and Mitchell A. Potter, "Evolving complex structures via cooperative coevolution," In *Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pp. 307-317, MIT Press, 1995.
19. M.-R Akbarzadeh-T, E. Tunstel, and M. Jamshidi, "Genetic algorithms and genetic programming: combining strengths in one evolutionary strategy," In *Proceedings of the 1997 Joint Conference on the Environment*, PP.373-377, Albuquerque, New Mexico, April 22-24, 1997
20. M.-R. Akbarzadeh-T., I. Mosavat, and S. Abbasi, "Friendship modeling for cooperative co-evolutionary fuzzy systems: a hybrid GA-GP algorithm," In *Proceedings of the 22nd International Conference of North American Fuzzy Information Processing Society*, pp.61-66, Chicago, Illinois, 2003.
21. M. Torabi-M, M.-R. Akbarzadeh-T., M. Khademi, "Learning ball interception skill : genetic algorithm vs. backpropagation," In *Proceedings of the National Computer Conference*, Ferdowsi University of Mashad, December 2002.
22. M. A. Potter & K. A. DeJong, "Cooperative coevolution: an architecture for evolving coadapted subcomponents," Evolutionary Computation, Vol. 8, No. 1, pp. 1-29, spring 2000.
23. D.E. Moriarty, "Symbolic evolution of neural networks in sequential decision tasks," The University of Texas at Austin, Jan 1997.
24. M. A. Potter, "The design and analysis of a computational model of cooperative coevolution," PhD Dissertation, George Mason University, Dept. of Computer Science, 1997.
25. M.N.H. Siddique and M.O. Tokhi, "Training neural networks: Backpropagation vs. Genetic algorithms," *IEEE Proceeding of Int. Conf. On Neural Networks*, Vol. 4, pp. 2673-2978, 2001.
26. M. Torabi-M, M.-R. Akbarzadeh-T., M. Khademi, " Structural Optimization of Perceptron Using Hybrid GA/BP Algorithem," In *Proceedings of the National Artificial Intelligence Conference*, Ferdowsi University of Mashad, October 2003.
27. S. Andre, E .Corten, K. Dorer, P. Qgugenberger, M. Joldos, J. Kummeneje, P. A. Navaratil, I. Noda, P. Riley, P. Stone, T. Takahashi & T. Yeap, " Soccer server manual," version 7.9 , Jan. 22, 2000.
28. P. Stone, M. Veloso & P. Riely, "A layered approach to learning client behaviors in Robocup Soccer Server," Applied Artificial Intelligence, No. 12, 1998.