Learning and Nonlinear Models – Revista da Sociedade Brasileira de Redes Neurais, Vol. 2, No. 2, pp. 73-83, 2004
© Sociedade Brasileira de Redes Neurais

PARALLEL GENETIC ALGORITHM WITH DIFFERENT EVOLUTION BEHAVIOR FOR MULTILAYER PERCEPTRONS DESIGN AND LEARNING

Ana Claudia M. L. Albuquerque, Jorge D. Melo, Adrião D. Dória Neto
Departamento de Engenharia de Computação e Automação
Universidade Federal do Rio Grande do Norte
Campus Universitário s/n - 59072-970 - Natal - RN – Brazil
aclaudia@dca.ufrn.br, jdmelo@dca.ufrn.br, adriao@dca.ufrn.br

Abstract – The combination of genetic algorithms and parallel processing can be very powerful when applied to the design and learning of Multilayer Perceptron neural networks. It is well known that the definition of Multilayer Perceptron neural networks is a very difficult task that could become, to a large extent, a process of trial and error. Also, the learning process of Multilayer Perceptron neural networks can be very slow, putting in danger the performance of countless applications. Therefore techniques for automating the design of neural networks are clearly of interest and the use of parallel processing is essential in minimizing the time required on the learning process. Furthermore, the use of cooperation in the genetic algorithm allows the interaction of different populations, avoiding local minima and helping in the search of the ideal solution. Finally, the use of exclusive evolution behavior on each copy of the genetic algorithm running in each task helped in enhancing the diversity of populations and the search for the fit solution.

Keywords – Neural networks, parallel processing, evolutionary computation, genetic algorithms.

Introduction

The Multilayer Perceptron networks have received growing attention due to their potential applications. The desirable characteristics such as versatility, simplicity, computational efficiency, accuracy, and high degree of applicability have motivated the use of these tools as a global interpolator and as a pattern classifier [1].

Over the years, the Multilayer Perceptrons have been proposed for the solution of several problems by means of the error back-propagation algorithm. The back-propagation algorithm is applied into the learning process of Multilayer Perceptron neural networks to refine the network synaptic weights. Even though the use of the back-propagation algorithm produced good results in refining the synaptic weights of the network, it is prone to becoming trapped in local minima and is incapable of optimization where problems lie in a non-differentiable problem space [2].

Interest in training neural networks using genetic algorithms has been growing rapidly in recent years [3]. Neural networks and genetic algorithms can be combined in a way that a population of neural networks competes against each other in a Darwinian setting. Each individual of the population represents a certain neural network with differing architecture and synaptic weights. The individuals codifying neural networks that produce good results are combined and passed onto the next generation. After a number of iterations, an optimized neural network can be obtained.

More recently, the idea of neuroevolution was used to search the space of neural networks architectures directly by using a genetic algorithm [4]. For neuroevolution to work, the environment doesn't need to satisfy any particular constraint. The only thing needed by a neuroevolution system is the existence of an effective way to evaluate the relative quality of candidate solutions. If the environment contains sufficient regularity for a task to be solvable, and the phenotype representations are sufficiently powerful, the neuroevolution can find a solution.

Different approaches of neuroevolution differ from each other in how the neural network specifications are encoded into strings. In neuroevolution, a chromosome can encode any relevant network parameter such as synaptic weights values, architecture of the network, number of hidden neurons and connectivity. The choices of encoding schemes are a significant role in shaping the search space, the behavior of the search algorithm and how the network genotypes are transformed into their phenotypes for evaluation.

Also, it is well known that, in general, the design of neural networks for specific applications is a very difficult task. Therefore, defining architectures of neural networks under a given set of constraints for specific situations can become a process of trial and error, depending, mostly, on past experiences with similar applications [5].

The great majority of learning algorithms designed to train neural networks, including the error back-propagation algorithm, are not able to determine an ideal architecture of neural network for a certain application. Instead, they only refine the network synaptic weights. Therefore, techniques for automating the design of neural networks are clearly of interest and a natural candidate for the application of evolutionary algorithms.

In this paper, a cooperative approach of genetic algorithms with different evolution behaviors is used to refine the synaptic weights and to define the architecture of the Multilayer Perceptron neural networks. Since the learning process of the Multilayer Perceptron neural networks can be very slow, varying according to the size of the network, parallel processing techniques was incorporated into the genetic algorithm developed. The parallelization of the genetic algorithm aims to minimize the time required on the training process and to improve the applications performance.

There are many ways to explore parallelism in genetic algorithms, as it will be introduced later. In [6], a competitive co-evolution model, where individuals selected from two independently evolving populations of hosts and parasites would directly compete against each other was explored. This approach of competitive co-evolution, however, can lead to an "arms race" with two populations fighting in search of the best solution.

In this paper, several parallel processes were created, each one of them corresponding to a different population. All populations created will evolve simultaneously. At the end of a predetermined number of times, the created processes will be able to communicate in order to exchange information concerning the best individuals selected by each one of them, so that the error signal produced by the neural network tends to zero and, consequently, it is considered trained. The exchange of information between the populations is very important, allowing them to cooperate and exploit promising areas of the search space found by other populations and, also, reintroduce in the population previously lost genetic material [7]. Furthermore, different evolution behaviors were introduced in each one of the coexisting populations. The use of distinct evolution behavior will contribute on the maintenance of the diversity of the individuals regarding each population. Applications in approximation of functions were developed and illustrate the effectiveness of the algorithm.

This paper is organized in the following way. In Section 1, it will be discussed parallelism in Multilayer Perceptron learning as well as in genetic algorithms. In Section 2, it will be shown how to train a Multilayer Perceptron with genetic algorithms. In Section 3, the details of the parallel genetic algorithm developed in this paper are given. Section 4 illustrates three applications in approximation of functions. Finally, Section 5 will present the conclusions obtained from the use of genetic algorithms and parallel processing into the design and learning of Multilayer Perceptrons neural networks.

1. Parallelism in Multilayer Perceptron Learning and Genetic Algorithms

Parallel processing techniques have been used in the learning process of the Multilayer Perceptron networks in order to minimize the amount of time consumed on the training process of more complex networks topologies and enlarge their range of applications. Several possibilities have been exploited in literature [8], [9]. In [10] a new parallel algorithm was presented, based on the cooperation concept. Multiple copies of the neural network in each task allow new parallel strategies. Information is periodically exchanged among the tasks to efficiently guide the search procedure and the back-propagation algorithm was used to minimize the error signal.

The concept of cooperative parallel algorithms differs from the traditional approach of parallel algorithm development [11]. Instead of dividing the computational load among the tasks and having them only to compete against each other, complete problems are solved in each task (competitive approach) and their solutions are speeded-up with the information provided by the other ones (cooperative approach).

The use of parallel genetic algorithms is a very good alternative since they are able to improve the performance of the genetic algorithms both in terms of velocity as in terms of enhancing the search quality since the maintenance of more diverse subpopulations helps to avoid premature convergence (local minima).

In genetic algorithms, parallelism was exploited in different levels producing both coarse and fine grain solutions. The fine-grained model assumes the placement of only one member of the population on each processing node. Therefore, the individuals can only reproduce and exchange genetic material with other individuals in a bounded region, as opposed to global ones. The coarse-grain model, on the other hand, is the most popular model used and assumes the division of a large population into several subpopulations with or without communication among the tasks [12]. Therefore, multiple processors run a sequential genetic algorithm on their population. Cooperation can be used in coarse-grain parallel genetic algorithms, where processors exchange individuals from their subpopulation with other processors. In the island model, individuals can randomly migrate from one population to another. In the stepping stone model, however, the individuals can migrate only to geographically nearby subpopulations. The existence of isolated subpopulations will help in the maintenance of genetic diversity. Individuals and evolution behavior can be exclusive on each copy of the genetic algorithm running in each task and communications can be used to accelerate the evolutionary process. In the following sections, this approach will be explored.

2. Training Multilayer Perceptrons with Genetic Algorithms

A genetic algorithm consists of a dynamic search method based on the theory of natural selection and can be successfully used in the learning process of a Multilayer Perceptron neural network [3], [13], [14], [15]. Each individual in the population has its genetic material represented by a finite string of binary symbols (chromosome). The chromosome of an individual will encode only one specific neural network. Therefore, in each population there are a finite number of individuals representing one neural network, with differing architecture, as well as synaptic weights. At the end of the training process, the selected individual will hold the final configuration of the neural network. The training process of Multilayer Perceptron consists of the refinement of the network synaptic weights. Simultaneous to the refinement of the synaptic weights, the network architecture is defined.

The genetic algorithm will proceed in the following way: an initial population of individuals is generated randomly. Each individual's genetic material, which consists of a vector of 0's and 1's, will contain a description of a specific neural network. Later, the individuals are decoded and evaluated according to a predefined fitness function. The best individuals are the ones capable to approximate the fitness function to zero. A function that converts a vector of 0's and 1's into a real number (double) was created to decode the information from each individual's chromosome.

The fitness function is calculated for each set of the training samples by adding all the square error signals obtained for each element located in the output of the neural network and dividing the result by the number of samples applied to the network. The error signal, meanwhile, is the result of the subtraction between a desired response previously defined and the actual response obtained by the network.

If the stop condition is not yet reached, the learning process continues and a certain number of individuals are chosen, according to the selection criteria, to be the parents of the next generation. In order to maintain a high degree of diversity among the individuals, different reproduction criteria were incorporated into the genetic algorithm.
In general, it can be said that to form a new population, individuals are selected according to their capability of minimizing the fitness function. Thus, the individuals that produced smaller error signals have a better chance of reproducing, while the others are more likely to disappear.

The main steps of the implementation of the genetic algorithm are illustrated in the fluxogram in Fig. 1.

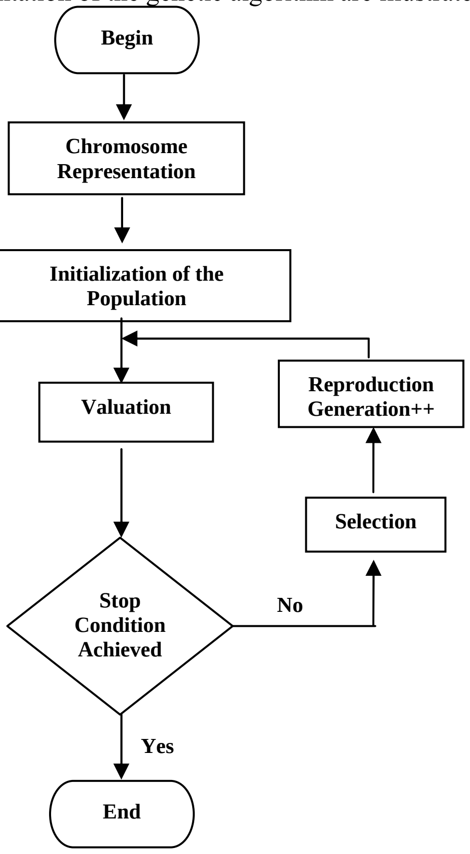
Begin
Chromosome Representation
Initialization of the Population
Valuation
Reproduction Generation++
Selection
Stop Condition Achieved
No
Yes
End

Fig. 1. The main steps of the implementation of the genetic algorithm

3. Cooperative Parallel Genetic Algorithm with Different Evolution Behaviors

The cooperative parallel genetic algorithm with different evolution behaviors developed in this paper is used to determine the Multilayer Perceptron neural network synaptic weights and architecture. In each population, there are a finite number of individuals codifying one specific Multilayer Perceptron neural network with differing architectures and synaptic weights. As the populations evolve, the network's synaptic weights are being refined along with its architecture. In the end, from the genetic material of the individual that best fit the solution, both architecture and set of synaptic weights for the neural network are extracted.

It is worth to stress that the definition of neural networks' architectures can become a process of trial and error, relying mostly on past experience with similar applications. Also, the performance of neural networks on a large number of applications is critically dependent on the choice of an ideal architecture. As a result, it is very hard to pre-define architectures for Multilayer Perceptron neural networks for a certain problem without previous knowledge or experience with similar applications. The use of genetic algorithms is, therefore, a natural and intuitive way to accomplish such task.

Besides defining the network's architecture, the genetic algorithm is used, simultaneously, to refine the network's synaptic weights. In order to enhance the search for the fit individual, different reproduction criteria were incorporated into the genetic algorithm. The use of different reproduction criteria will contribute on the maintenance of the diversity of the individuals regarding each population, speeding up the search for the ideal solution. Therefore, existing populations will evolve differently from one another following its own criterion of reproduction.

The reproduction step of a genetic algorithm can be performed using many different kinds of heuristics. In the algorithm developed, the individuals are ordered according to value of the fitness function produced by them. The quantity of individuals that will become parents of the next generation is, then, chosen randomly. Note that the parents of the next generation are the individuals that produced the lowest error signals. From the set of individuals chosen to be parents of the next generation, two are randomly chosen to have its genetic material combined, producing an offspring. The genetic material can be combined in one or two different points. A small number of mutations, also obtained in a random manner, were introduced to the new population. Fig. 2 illustrates two examples of reproduction of the genetic algorithm.

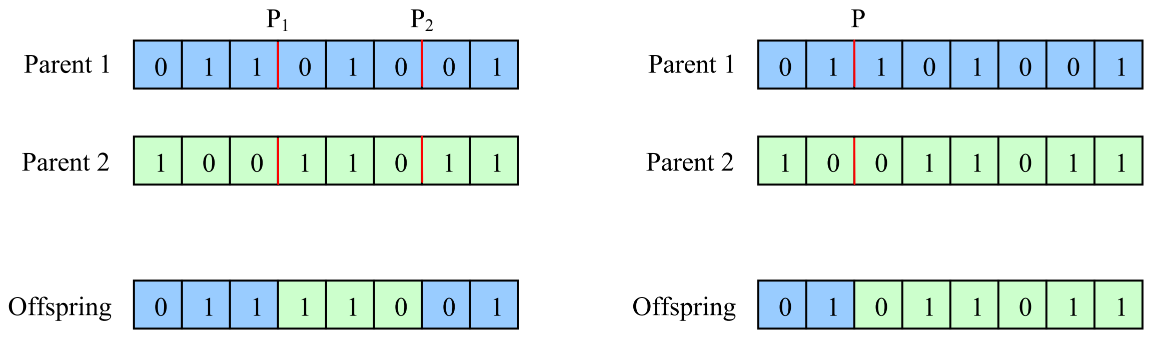

Fig. 2: Illustration of two examples of reproduction of the genetic algorithm

Besides presenting different evolution behaviors, the existing populations are able to communicate with each other and exchange valuable information on the best individual selected by each one of them so far. The presence of cooperation in the genetic algorithm is, therefore, very important since the exchange of information between the populations helps to avoid local minima. Also, it allows the exploitation of a larger range of the search space and reintroduces previously lost genetic material.

The genetic material of each individual contains two main fields. The first field codifies an index to a table of architectures. The second field codifies the synaptic weights for the architecture defined in the first field. Each synaptic weight is represented by a 32-bits binary string. The index to a table of architectures is also represented by a 32-bits binary string. During the initialization, the binary strings representing the network's architecture and synaptic weights are generated randomly. As the populations evolve, the best architectures and set of synaptic weights are maintained. Finally, the best individual will hold the final configuration of the neural network. Fig. 3 illustrates the representation of the genetic material of each individual. Note that the table of architectures will contain different kinds of networks with different numbers of layers, as well as neuron per layer. However, the input and output dimensions of the neural network are pre-determined since they depend and vary according to the application.

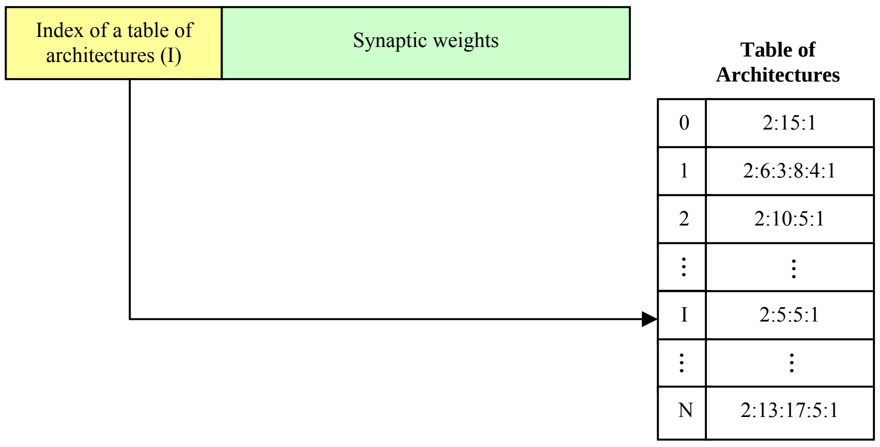

Fig. 3: Representation of the genetic material of an individual

Once again, each individual of the population codifies different neural networks' architectures. Therefore, the size of the genetic material of each individual varies according to the size of the neural network it represents, i.e., an individual codifying a simpler neural network will present a smaller set of synaptic weights when compared to an individual that codifies a much more complex architecture.

However, it is necessary to standardize the representation of the genetic materials since, performing operations on differently sized chromosomes, memory that is not part of the smaller chromosome will be used in the recombination process. The simpler solution to this problem consists on taking as standard the size of the genetic material of the individual that codifies the largest neural network. As a result, all the individuals would present chromosomes with same sizes and the layers, neurons and connections that do not exist would be represented by zero. However, by doing so, the size of the neural networks and the genetic materials could increase inappropriately. Besides demanding a lot of memory and making the recombination process very slow, this solution increases the amount of time required on the exchange of messages performed by the parallel tasks, which can badly affect the final performance of the algorithm.

The most viable solution found was based on the specification of a reasonable interval of synaptic weights for the networks. Therefore, all possible neural networks' architectures that obeyed the specified interval were considered. By doing so, the uncontrolled growth of the genetic materials can be avoided. Furthermore, the massive use of memory space is also avoided and, more importantly, the message exchange among parallel tasks is accelerated. Therefore, during the initialization of the populations, the amount of individuals at each population and the interval of synaptic weights are previously specified. Later, the genetic material of each individual is generated randomly.

The parallel structure adopted for the implementation of the parallel genetic algorithm is illustrated in Fig. 4.

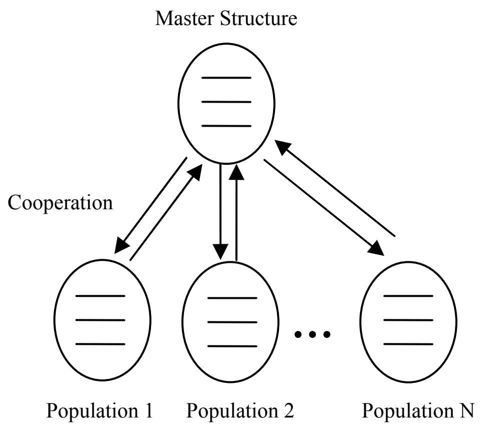

Fig. 4. The parallel structure adopted in the genetic algorithm

As can be seen through Fig. 4, there is a master structure that is responsible for the initialization of the populations. Each population will evolve simultaneously, following its own reproduction criteria. After a predetermined number of generations, each one of the populations will send to the master structure the best individual – i.e. the individual that has produced the lowest error signal. From this set of best individuals received from all the coexisting populations, the master structure will select the one with the lowest error signal. Finally the master structure will send a copy of the individual selected to each one of the coexisting populations and so on, until the error signal tends to zero.

The environment used for the implementation of the parallel algorithm was the PVM (Parallel Virtual Machine). The software PVM is an environment for parallel and distributed computation. It allows the user to create and to access a parallel computation system made up from a collection of distributed processes, as well as to treat the resultant system as a unique virtual machine, hence the name parallel virtual machine. The software PVM is based on the message-exchange parallel programming model. In this way, messages are exchanged among the tasks through a connection chain.

4. Application on Approximation of Functions

The genetic algorithm developed in this paper was used in the learning process of the Multilayer Perceptron network in order to approximate functions. Two functions, $z = sin(r)/r$ and $f(x1, x2) = cos(2\pi x1)\cdot cos(2\pi x2)$, were used.

As was said before, the great advantage of the cooperative parallel genetic algorithm is the concept of having different populations with different evolution behaviors evolving simultaneously. To illustrate that, it is drawn a comparison between the sequential and the parallel approach of genetic algorithms.

The first function used was the following: $z = sin(r)/r$, where $r = sqrt(x2+y2)$ and $-7.5 \leq x \leq 7.5$ and $-7.5 \leq y \leq 7.5$. Initially, a sequential genetic algorithm was used on the approximation of the function. Thus, there are five populations evolving independently, without any degree of cooperation neither communication. The new generations are born from the recombination of individuals that belong to each one of the five populations only.

The mean square error (MSE) signal of the five populations is illustrated on Fig. 5. However, it can also be noticed the existence of populations, such as 1 and 3, presenting higher error signals that, due to the lack of cooperation, are not helped by the others. The performance of the neural network can, then, be affected in a bad way.

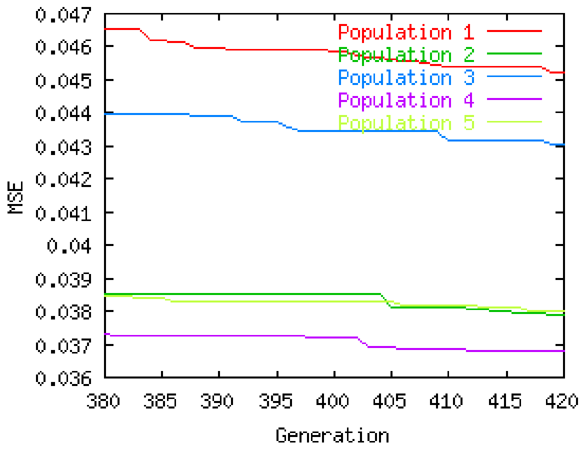

Fig. 5: Mean square error signal of the sequential genetic algorithm for the function $z = sin(r)/r$

Later, the cooperative parallel genetic algorithm with different evolution behaviors was used on the approximation of the function $z = sin(r)/r$. The use of parallel processing, along with cooperation and different evolution behaviors helped to accelerate the convergence of the neural network.

In Fig. 6, there are five populations evolving simultaneously, where each line represents the best individual produced so far by each one of them.

Right from the start, it can be noticed, through Fig. 6, that the error signals of populations 1, 2, 3 and 5 were brought down to the same level of the Population 4 at the 360th generation. Then, all the existing populations went on evolving simultaneously, competing and cooperating with one another every once in a while. Finally, from the help of its neighbors, the Population 2, which started out with the second highest error signal, accomplished the lowest one in the end, at generation 420.

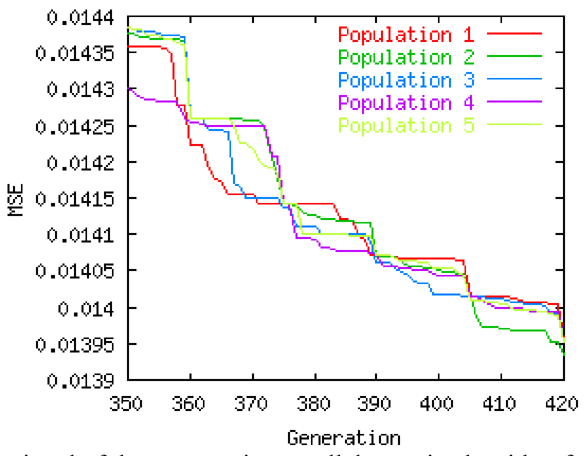

Fig. 6: Mean square error signal of the cooperative parallel genetic algorithm for the function *z = sin(r)/r*

The parallel genetic algorithm developed was used to define the Multilayer Perceptron neural network architecture. Several configurations of neural networks were used. The number of hidden layers varied from one to six and the number of neurons per hidden layer varied from three to fifteen. There were two neurons in the input layer and one neuron in the output layer. The number of neurons per input layer and the number of neurons per output layer were pre-determined since they depend on the application.

The final configuration of the Multilayer Perceptron neural network presented two neurons in the input layer, one neuron in the output layer and two hidden layers with ten neurons in the first hidden layer and six neurons in the second.
The reconstructed output of the function *z = sin(r)/r* is illustrated in Fig. 7. In Fig. 8, it is presented the original output of the function. Finally, a superposition of both outputs is shown in Fig. 9.

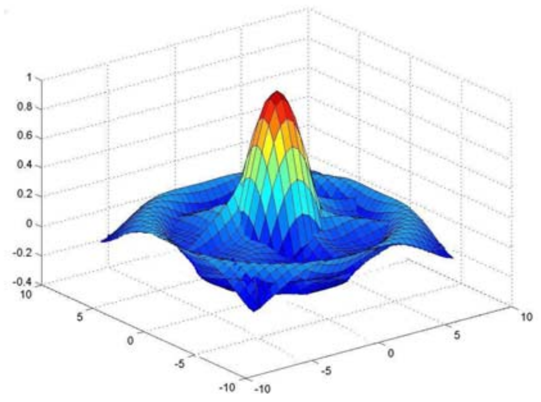

Fig. 7: Reconstructed output of the function *z = sin(r)/r* obtained from the Multilayer Perceptron

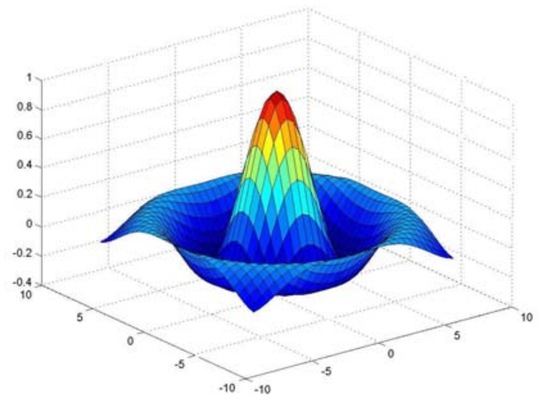

Fig. 8: Original output of the function *z = sin(r)/r*

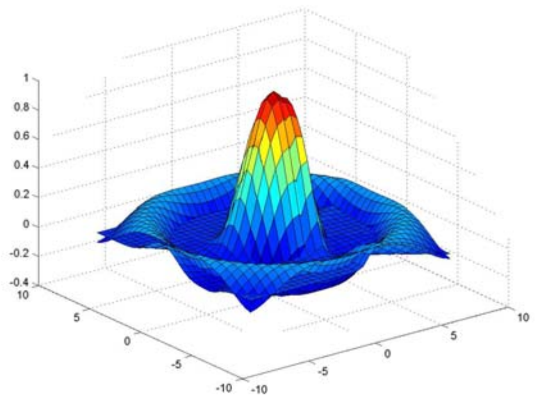

Fig. 9: Superposition of the original and reconstructed outputs for the function *z = sin(r)/r*

Finally, the Multilayer Perceptron neural network was used to approximate the function *f(x1, x2) = cos(2πx1)· cos(2πx2)*.
First, a sequential genetic algorithm is used and the MSE signal of the five populations evolving independently is illustrated on Fig. 10.

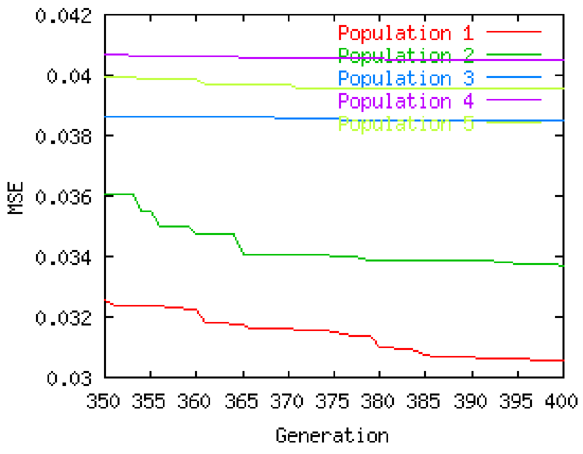

Fig. 10: Mean square error signal of the sequential genetic algorithm with for the function *f(x1, x2) = cos(2πx1)· cos(2πx2)*

All five populations evolve independently, without exchange of information or cooperation among them. Once again, with this approach, is very easy to find populations stuck in local minima, such as Populations 3, 4 and 5.
On the contrary, through Fig. 11, where there are five populations evolving simultaneously, it is possible to realize the improvement that the cooperative parallel genetic algorithm with different evolution behavior represents.

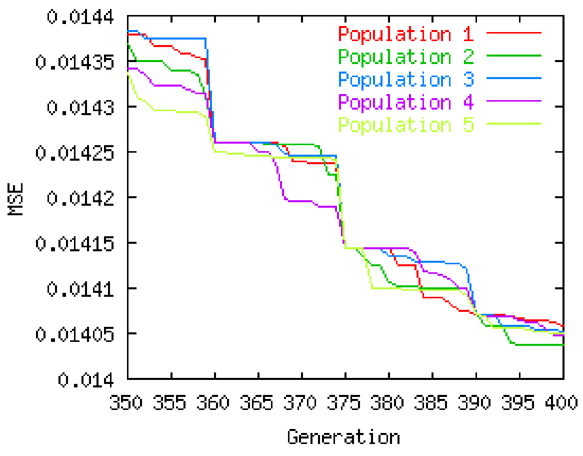

Fig. 11: Mean square error signal of the cooperative parallel genetic algorithm for the function *f(x1, x2) = cos(2πx1)· cos(2πx2)*

As can be seen through Fig. 11, from the 360th generation up to the 375th, the members of Population 4 were the ones with the lowest error signal. However, at the 375th generation, due to the exchange of genetic material among the populations, the error signal produced by Populations 1,2,3 and 5 were brought down to the same level of Population 4.

Once again, the parallel genetic algorithm developed was also used to define the Multilayer Perceptron neural network architecture. Several configurations of neural networks were used. The number of hidden layers varied from one to six and the number of neurons per hidden layer varied from three to fifteen. There were two neurons in the input layer and one neuron in the output layer. The number of neurons per input layer and the number of neurons per output layer were pre-determined since they vary according to the application.

The final configuration of the Multilayer Perceptron neural network presented two neurons in the input layer, one neuron in the output layer and two hidden layers with six neurons in the first hidden layer and eight neurons in the second.

Therefore, the advantage of the cooperative parallel genetic algorithm with different evolution behaviors, besides allowing the search of the ideal architecture for network given a certain application, is having several populations evolving simultaneously. This evolution will take place independently, up to a given position in which all populations cooperate with one another by the exchange of genetic material. Up to this point, all populations will have the fit individual so far produced. Therefore, the populations that were captured in high errors signals immediately pass to the same level of the others, avoiding local minima. In other words, while the existence of differing evolution behaviors helps maintain the diversity, the use of cooperation gives the opportunity to all the populations to evolve in such a way that they will become the best one.

The reconstructed output of the function *f(x1, x2) = cos(2πx1)· cos(2πx2)* is illustrated in Fig. 12. In Fig. 13, it is presented the original output of the function. Finally, a superposition of both outputs is shown in Fig. 14.

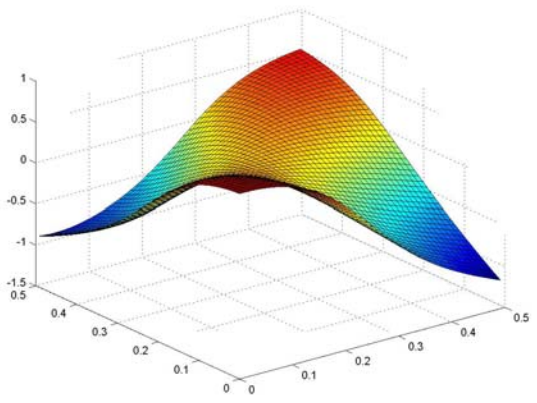

Fig. 12: Reconstructed output of the function *f(x1, x2) = cos(2πx1)· cos(2πx2)* obtained from the Multilayer Perceptron

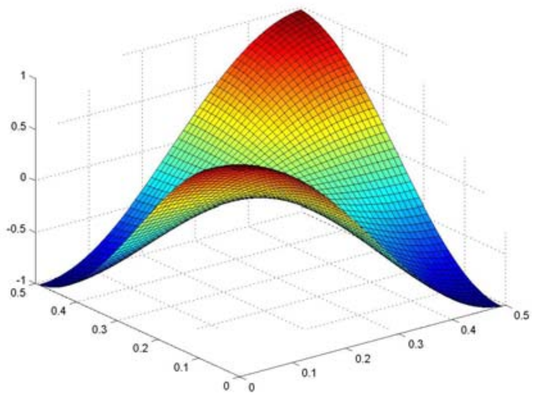

Fig. 13: Original output of the function *f(x1, x2) = cos(2πx1)· cos(2πx2)*

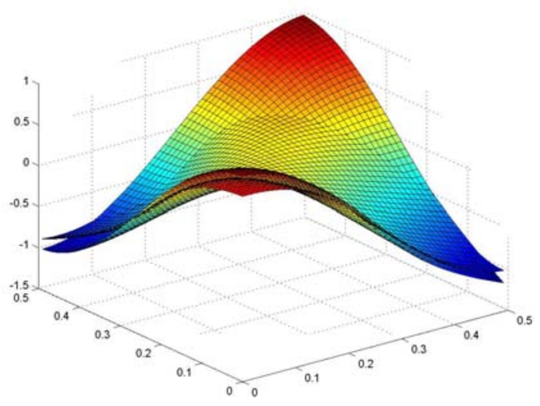

Fig. 14: Superposition of the original and reconstructed outputs for the function *f(x1, x2) = cos(2πx1)· cos(2πx2)*

5. Conclusions

The cooperative parallel genetic algorithm with different evolution behaviors used in the learning process of the Multilayer Perceptron neural network was very efficient when compared to the sequential form of the genetic algorithm. The cooperative parallel genetic algorithm was applied to the approximation of functions but can have its use extended to the several other kinds of applications with neural networks.

As it could be seen through the analysis of the obtained results, the simple fact of allowing the exchange of information among different populations represented a great achievement in the final performance of the algorithm. Therefore, through cooperation, different populations could interact within each other, avoiding local minima and helping in the search of the ideal solution.

Thus, the concept of having a pure competitive algorithm, where populations will only compete against each other in order to find the best individual was modified by the insertion of cooperation between populations. In this way, starting from a certain position, the different populations will contain the best individual up to now found. Therefore, populations associated to high error signals immediately drop to the same level of the others.

Furthermore, it is worth to stress that the use of different evolutionary behaviors in each population enabled a larger diversification of them, speeding up the search for the ideal solution as well.

Also, the genetic algorithm was used simultaneously to refine the network's synaptic weights and to define its architecture. In general, defining architectures for neural networks is a very hard task and most of the learning algorithms developed only refine their synaptic weights. Therefore, the combination and application of evolutionary computation into the design of neural networks are clearly of interest.

6. References

[1] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Maxwell Macmillan International, 1994.

[2] D. Curran, C. O'Riordan, "Applying Evolutionary Computation to Designing Neural Networks: A Study of the State of the Art", *Technical Report: NUIG-IT 111002, 12 pages*, National University of Ireland, Galway, Ireland, October 2002.

[3] M. Siddique, M. Tokhi, "Training Neural Networks: Backpropagation vs Genetic Algorithms", *INNS-IEEE International Joint Conference on Neural Networks*, Washington DC, 14-19, July 2001.

[4] F. Gomez, *Robust Non-linear Control through Neuroevolution*, PhD Thesis, Artificial Intelligence Laboratory, University of Texas at Austin, August 2003.

[5] K. Balakrishnan, V. Honavar, "Evolutionary Design of Neural Architectures", *Technical Report ISU CS-TR 95-01*, Iowa State University, January 1995.

[6] M. Potter, K. DeJong, "Cooperative coevolution: An architecture for evolving coadapted subcomponents", *Evolutionary Computation*, 8(1):1--29, 2000.

[7] P. Adamidis, S. Kazarlis, V. Petridis, "Advanced Methods for Evolutionary Optimisation", *LSS'98, 8th IFAC/IFORS/IMACS/IFIP Symposium on Large Scale Systems: Theory and Applications*, University of Patras, Greece, July 15 -17, 1998.

[8] J. Torresen, *Parallelization of Back-propagation Training for Feed-Forward Neural Networks*, PhD Thesis, The Norwegian Institute of Technology, 1996.

[9] S. Foo, P. Saratchandran, N. Sundararajan, "Parallel Implementation of Back-propagation Neural Networks on a Heterogeneous Array of Transputers", *IEEE Trans. Systems, Man and Cybernetics - Part B*, Vol. 27:1, pp. 118-126, 1997.

[10] R. Alves, J. Melo, A. Dória Neto, A. Albuquerque, "New Parallel Algorithms for Back-Propagation Learning", *INNS-IEEE International Joint Conference on Neural Networks*, Honolulu, USA, 2002.

[11] T. G. Crainic , M. Gendreau, "Cooperative Parallel Tabu Search for Capacitated Network Design", *Journal of Heuristics*, 8, 601-627, 2002.

[12] A. Muhammad, A. Bargiela, G. King, "Fine-grained Parallel Genetic Algorithm: A Global Convergence Criterion", *Int. Journal of Computer Mathematics*, Vol. 73(2), 139-155, 1999.

[13] Z. Michalewicz, *Genetic Algorithms+Data Structures=Evolution Programs*, Springer-Verlag, 1992.

[14] J. Schaffer, D. Whitley, L. Eshelman, "Combinations of Genetic Algorithms and Neural Networks: A Survey of the State of the Art", *Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN-92)*, Baltimore, USA, 1992.

[15] D. Foster, J. McCullagh, T. Whitfort, "Evolution Versus Training: An Investigation into Combining Genetic Algorithms and Neural Networks", *Proceedings of the International Conference on Neural Information Processing (ICONIP-99)*, Perth, Australia, 16-20 November 1999.