Hierarchical Evolution of Heterogeneous Neural Networks

Daniel Weingaertner, Victor K. Tatai, Ricardo R. Gudwin, Fernando J. Von Zuben
DCA – FEEC – State University of Campinas (UNICAMP)
Po.Box 6101 – CEP 13083-970 – Campinas, SP – Brazil
{danielw, tatai, gudwin, vonzuben}@dca.fee.unicamp.br

Abstract – This paper describes a hierarchical evolutionary technique developed to design and train feedforward neural networks with different activation functions on their hidden layer neurons (Heterogeneous Neural Networks). At an upper level, a genetic algorithm is used to determine the number of neurons in the hidden layer and the type of the activation function of those neurons. At a second level, neural nets compete against each other across generations so that the nets with the lowest test errors survive. Finally, on a third level, a coevolutionary approach is used to train each of the created networks by adjusting both the weights of the hidden layer neurons and the parameters for their activation functions.

I. INTRODUCTION

Multi-Layer Perceptron (MLP) and Radial Basis Function (RBF) networks are the most common feedforward artificial neural networks (ANNs) used on classification, recognition and prediction problems. They have been applied to a variety of fields, usually producing better results when replacing or being combined with conventional techniques. Such capabilities come from the fact that both MLPs and RBFs show an important attribute: they are universal approximators with good generalization capabilities [11], [2].

Typically, ANNs are composed of a pre-defined number of similar neurons in their hidden layer. Despite the existence of some rules of thumb to indicate this parameter in accordance with the available training data, such empirical methods are failure-prone, not warranting their applicability in all cases. In such context, designing ANNs through simulated evolution has shown its effectiveness as an automatic alternative to manual configuration [5]. A prominent advantage of the evolutionary design over the manual one is its adaptability to dynamic changes in the environment [16].

Although the neuron's activation function (AF) has been shown to be an important parameter for the ANN configuration — it is the source of non-linearity responsible for the approximation capabilities — few works have explored such fact while devising new neural net topologies. Currently, there are no extensive results reported in the literature contemplating the effectiveness of employing gaussian or logistic AF's in all neurons of the hidden layer. This approach is followed, generally, as a means to simplify the design tasks.

However, if the AF of each neuron could be automatically defined, better results regarding the net performance and the training convergence rate could be achieved. In this work, we attempt to investigate such supposition by using Genetic Algorithms (GAs) for the design of heterogeneous neural networks. The idea is to view each hidden neuron as a peer with distinguished capabilities (different AF) to be explored while integrating with others in a same ensemble (neural net). Such abstraction is based on the cooperative combination of evolving populations of experts, combining their qualifications to reach the problem solution [2].

With such objective in mind, we used a hierarchical approach, known as Hierarchical Coevolutionary Genetic Algorithm (HCGA), which was initially conceived with two evolutionary levels [3]. We added one more evolutionary level to it, so that on the first level a GA is used to build the ANN's topology, i.e., to choose both the number of neurons in the hidden layer and the AF of each neuron. On a second level, neural nets compete against each other across generations so that the nets with the lowest test error survive. Finally, on the third level, a coevolutionary approach is used to train each of the above-created networks by adjusting the weights of the neurons in the hidden layer and the parameters of their AF's.

Thus we propose an algorithm capable of constructing efficient ANNs without the need of expertise knowledge about the problem nor adjustment of configuration parameters. The resulting ANNs may have any number of neurons on their hidden layer as well as different AFs on those neurons, optimized in order to solve complementary parts of the problem.

In the remainder of the paper, we introduce the use of evolutionary techniques for the configuration of neural nets, present our framework to the automatic design of heterogeneous neural networks, show results from our experiments, compare this approach to a previous version [1], identify possible work extensions, and address some final considerations.

II. EVOLUTIONARY ALGORITHMS APPLIED TO NEURAL NETWORKS

GA's have been used in conjunction with neural networks in three major undertakings [5]: data pre-processing, weight optimization, and neural net topology determination. The second item has been hampered mainly because of the *Competing Conventions Problem*. As weight adjustment with GA's relies heavily on recombination, there may be many equivalent symmetric solutions to the same optimization problem, delaying the convergence process. This can be alleviated by using more appropriate crossover operators which try to avoid individuals presenting the same cyclic genetic order on the elements of their chromosomes.

Another strategy centers upon the integration between evolutionary programming (EP) and ANNs. Liu and Yao [16] have presented an EP-based algorithm for the tuning of ANNs with different activation function nodes. The weights are adjusted by means of a combination of the Backpropagation (BP) algorithm with a random search algorithm. For simplicity, the authors chose to use only the logistic and the Gaussian AF's, as they represent two broad classes of activa-

0-7803-7282-4/02/$10.00 ©2002 IEEE

tion functions with complementary features. The resulting *generalized neural net* (GNN), as they call it, resembles very much what we name here as heterogeneous neural networks (HNN), albeit our approach gears towards the employment of a hierarchical coevolutionary-genetic algorithm (HCGA) and is open to other types of AF's.

Iyoda and Von Zuben [6] have also attempted to analyze the impact of configuring ANNs with different AFs at the hidden layer by proposing an evolutionary hybrid architecture inspired by another constructive method (Projection Pursuit Learning - PPL). Such approach also incorporates distinct composition functions in the output layer (additive and/or multiplicative), leading to a higher efficiency on the combination of the mapping efforts realized by the hidden neurons. Their algorithm relies only on a classical GA, not directly promoting the cooperation among the units.

Finally, it is worth mentioning the prominent evolutionary methodology developed by Moriarty and Miikkulainen [3], [4], denominated SANE. SANE relates to a "symbiotic adaptive neuro-evolution system" in which a population of homogeneous neurons is coevolved to compose a neural net intended to be deployed on dynamic environments. This hierarchical solution attempts to optimize the neural topology by two means. First, since neurons are recognized as functional building blocks, their ensemble can be more accurately evaluated. Second, since no neuron is evaluated only by its own capabilities, but rather by the qualities of the groups in which it takes part, evolutionary pressure exists to evolve several complementary neuron types.

III. THE HCGA2 APPROACH

In this section, we present the main design decisions taken during the conception of the $HCGA_2$ approach and assess its adequacy and performance through a series of benchmarking results over well-known pattern classification problems.

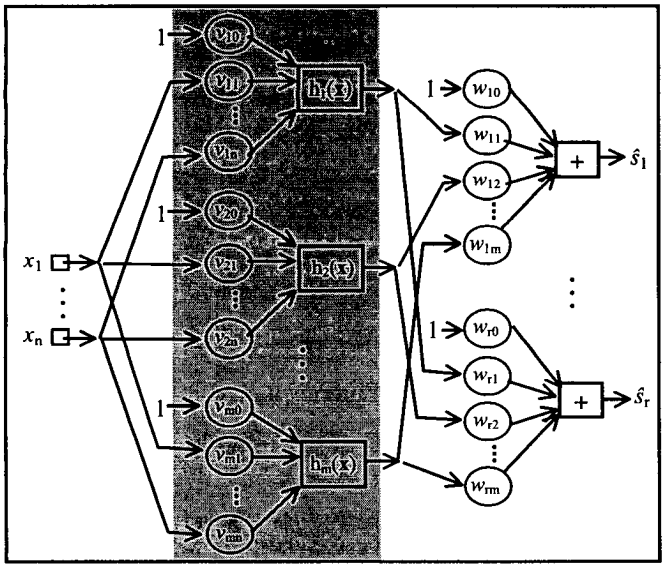

Figure 1: Architecture of the heterogeneous neural network to be evolved by the hierarchical approach. The shadowed area indicates the components to be optimized by the coevolutionary process: neurons with different activation functions and the associated weights.

A. Heterogeneous Neural Networks

Figure I shows the heterogeneous neural network (HNN) model used on the $HCGA_2$. It resembles a typical feedforward neural net; the main difference lies on the hidden layer, as its neurons may have distinct AF's ($h_j(\cdot)$). Such functions are chosen from a delimited candidate set, which is presented in Table I, and comprehends a broad parcel of those usually employed on the construction of ANN's.

It is important to emphasize that there is also a Radial Basis Function (RBF-Green [13]) that can be used together with the non-RBF functions, in a novel approach that we expect to increase the HNN's approximation capacity. On the RBF-Green function we can adjust up to three parameters: μ - the vector with the RBF's center coordinates; σ - the RBF's variance vector; and $\boldsymbol{w}$ – a pondering weights vector. To reduce the GA's search space, we only evolve the μ values and one value for σ. No pondering weights are used.

TABLE I
SET OF HIDDEN NEURONS ACTIVATION FUNCTIONS

Name	Activation Function
Linear	$h = x$
Signal Step	$h = \begin{cases} -1 \text{ if } x < 0 \\ +1 \text{ if } x > 0 \\ lastValue \text{ if } x = 0 \end{cases}$
Hyperbolic Tangent	$h = \frac{e^{bx} - e^{-bx}}{e^{bx} + e^{-bx}}$
Gaussian	$h = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-x^2}{2\sigma^2}}$
RBF-Green	$h = \sum_{i=1}^{n} w_i \, e^{\frac{1}{2\sigma_i^2}\|x-\mu_i\|^2}$

As illustrated (Figure I), $\mathbf{x}$ and $\hat{\mathbf{s}}$ represent, respectively, the input vector of the training/testing patterns and the resulting net output. The output neurons have all an additive linear combination as aggregation function. There are two sets of weighted connections, namely, $\mathbf{V}$ and $\mathbf{W}$. The later is optimized during a supervised configuration process, through stimulus-response pairs, by means of the Least Mean Square (LMS) method. The adjustment of the weights in $\mathbf{W}$ is defined in order to solve the following optimization problem:

$$\min_{W} \|y - s\|^2, \tag{1}$$

where $\|\cdot\|$ is the Euclidian norm and

$$y = HW = \begin{bmatrix} \mathbf{h}_1(x_1) & \mathbf{h}_2(x_1) & \dots & \mathbf{h}_m(x_1) \\ \mathbf{h}_1(x_2) & \mathbf{h}_2(x_2) & \dots & \mathbf{h}_m(x_2) \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{h}_1(x_p) & \mathbf{h}_2(x_p) & \dots & \mathbf{h}_m(x_p) \end{bmatrix} W \tag{2}$$

Given the training set $\{(\mathbf{x}_i, \mathbf{s}_i)\}_{i=1}^{p}$, the LMS will then try to minimize the sum of the squared errors produced by each of the p input-output patterns. The $\mathbf{H}$ matrix is obtained, in this

case, after the definition of m transfer functions chosen among the candidates given by Table I and applied to the p patterns thereupon. The optimal solution for the output weights is given by

$$H^T HW = H^T s \Rightarrow W = (H^T H)^{-1} H^T s, \quad (3)$$

where $(H^T H)^{-1} H^T$ is the *pseudo-inverse* of H, which shall only exist if H has a non-deficient rank.

B. $HCGA_2$ Architecture

The shadowed box in Figure I indicates the components of the HNN to be optimized directly by the $HCGA_2$. As depicted in Figure II, we conceived a hierarchical coevolutionary architecture composed of three levels. In an upper level, a conventional GA (NNdGA) is used to build the neural network description (NNd) or net topology, i.e., to choose the number of neurons in the hidden layer and their AFs (possibly distinct). Each gene in the chromosome represents one AF from Table I. Similar NNd's are not allowed, in order to reduce the possibility of equivalent symmetric solutions (*Competing Conventions Problem*).

Associated to each NNd, there is a second level GA (HNNGA) whose population consists of HNN's with the hidden layer built according to the NNd. Those HNN's take their hidden layer neurons at random from the bottom level GAs (NeuronGAs). Neural nets compete across generations and are refined using the iterated LMS optimization process applied to their output weights. The fitness of the HNNs is inversely proportional to the Root Mean Squared Error (RMSE), that is, corresponds to the percentage of correctly classified patterns. The NNd receives the fitness of the best ranked HNN of its HNNGA.

At the bottom level, a coevolutionary approach was selected to train the networks by adjusting both the AF parameters of the hidden layer neurons and the input weights. In order to promote the cooperative behavior among the neurons that participate in a given net (aiming at the optimized sharing and division of responsibilities), each unity receives its fitness according to the average fitness of the networks wherein it engages. Each neuron on the hidden layer can be viewed as a distinct evolving species that is allotted to a separate GA (NeuronGA) customized to represent its attributes and that will probably specialize in a complementary manner to the other NeuronGAs.

The chromosomes associated with the HNNs are codified as follows: each gene represents a logical link to a given neuron pertaining to a certain bottom level NeuronGA. The same neuron is allowed to integrate various neural nets. All HNN's in a HNNGA have the same architecture (since they were built after the same NNd) and differ by the hidden layer neurons they take from the NeuronGAs. Typically, the codification of the neuron chromosomes in the NeuronGAs comprehend two slots: one for the input weights and another for the neuron's AF parameters. The mutation and crossover operators may actuate on both slots at the same time, and they know how to manage the differences between them (scale, data type, etc.).

The evolution of the HNN's architecture and of the hidden layer weights are alternated. This process can avoid the moving target problem resulting from the simultaneous evolution of both architectures and weights [15]. The number of generations considered for benchmarking purposes relates to the NNdGA cycles. The initial populations of all GA's (NNdGA, HNNGA and NeuronGA) are set in a random manner (zero-average and uniform distribution). Configuration parameters can be found on Table II and Appendix A is dedicated to the $HCGA_2$ algorithm.

C. Results

The two pattern classification problems considered here were obtained from the PROBEN1 benchmarking repository [9], allowing the comparison of our proposal with others. The database of patterns in this repository is, for each problem, divided into three different partitionings of the same data set. This allows network simulation with the same set but with patterns in a different order each time. The data sets are

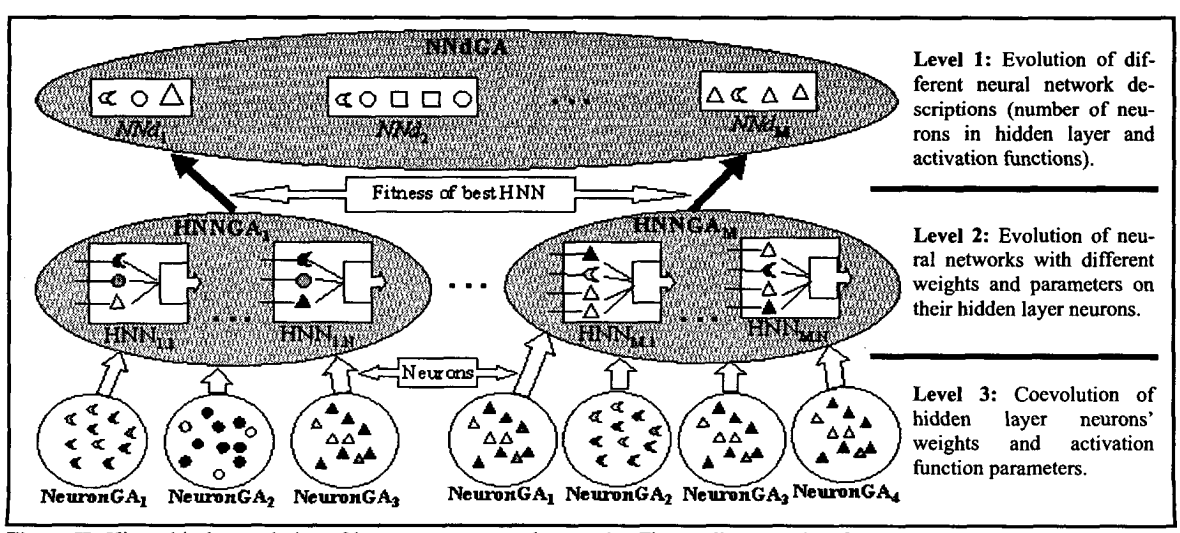

Figure II: Hierarchical coevolution of heterogeneous neural networks. The small geometrical forms represent neurons with different AFs. The NeuronGAs provide the neurons to the HNNs. The fitness of the best HNN is set on the corresponding NNd.

TABLE II
SYSTEM CONFIGURATION PARAMETERS

Parameter	Meaning	NNdGA	HNNGA	NeuronGA
MAX_GENERATIONS	Maximum number of generations	200	5	1
POPULATION_SIZE	Number of elements in the GA	20	30	30
RANDOM_PERCENT	Percentage of new randomly initialized elements to be inserted in the population	20%	20%	20%
ELITIST_SELECTION	Percentage of best elements copied to next generation (elitist selection)	10%	10%	10%
CROSSOVER_CHANCE	Percentage of elements to be created by crossover between the best elements	25%	25%	25%
MUTATION_CHANCE	Chance of mutation occurrence	1%	1%	1%
MIN_HIDDEN_NEURONS	Minimum number of neurons allowed in the net hidden layer	5	--	--
MAX_HIDDEN_NEURONS	Maximum number of neurons allowed in the net hidden layer	15	--	--
MAX_UNUSED_CYCLES	Maximum number of cycles a neuron may survive without belonging to any network	--	--	3

divided into training, validation and test patterns. The training patterns were used in the network refinement through the LMS process; the validation patterns were used to compute the HNN's fitness, and the test patterns were used to test the best HNN at the end of each NNdGA cycle. It is important to remark that the test result is not used by the $HCGA_2$ (not even to indicate when to stop the training phase, as done in many ANN training algorithms), thus working as an independent result.

The first problem, *Card*, refers to the task of approving or not the delivery of credit cards to particular customers, taking into account their profiles. The database has 690 samples of possible customers (patterns), with 51 input parameters (net input attributes) and two possible output responses (yes or no). 44% of the customers have good profiles and there are some absent data referring to some attributes, hindering the classification process. There are 345 training patterns, 172 validation patterns and 172 test patterns.

The *Heart* problem lies in the area of predicting cardiac diseases by observing some clinical cases (patients' health conditions). In this repository, there is a collection of 920 patient samples (patterns) each composed of 35 inputs (personal data) and two outputs (prone to cardiac problems or not). Some attributes are also missing, making the decision even harder. The sets are divided into 460 training patterns, 230 validation patterns and 230 test patterns.

The experiments were implemented in the Java language (JDK 1.2) and, on an IBM 9076SP (with four IBM/RS6000 Power2 Super Chip 160 MHz processors), it took about 25 min. to process each NNdGA cycle, confirming the high computational costs we expected. Most of the computational time was spent on the optimization of the output layer weights (using the LMS), more specifically, on computing the pseudo-inverse matrix, which is done about 4000 times at each cycle of the first level (NNdGA). But after obtaining the evolved net, no extra computational costs are present. Besides, the ease of use is a real advantage, since the designer does not need to know which AF's to use, how many neurons to put in the hidden layer, etc.

Each experiment was run five times. Table III shows the mean percentage (μ) and standard deviation (σ) of the participation of each AF type in the best networks evolved for each problem. C# stands for the *Card* data set and H# for the *Heart* data set. As can be noted, some types of neurons appeared more frequently than others, but not in a significant manner that would allow us, for example, to caracterize a tendency for these classification problems.

By observing the fitness of the best evolved network along the generations we notice that it does not converge too fast, indicating that our HNN population is able to maintain diversity and escape from *local maxima* even though we use an elitist selection approach. The average fitness of all networks does also not oscillate much, because the changes in the neurons during their evolution is smooth, due to the use of the geometrical crossover [17] and inductive mutation [14]. The test data correct classification percentage, despite some big punctual oscillations, does not change significantly, indicating that the best network does not loose much of its generalization capacity.

TABLE III
PERCENTAGE OF NEURONS OF EACH TYPE ON THE BEST NETWORK, FOR EACH DATA SET.

		Linear	Signal Step	Hyperbolic Tangent	Gaussian	RBF-Green
C1	μ	18.92%	14.95%	13.52%	27.14%	25.43%
	σ	8.77	7.59	10.55	5.59	20.66
C2	μ	25.23%	19.90%	16.82%	20.10%	17.95%
	σ	7.46	13.914	9.90	9.90	12.66
C3	μ	21.80%	14.14%	18.06%	24.64%	21.36%
	σ	21.32	13.68	14.52	15.72	14.26
H1	μ	12.82%	9.95%	20.43%	23.29%	33.50%
	σ	10.92	10.41	17.40	18.29	7.33
H2	μ	18.30%	21.28%	20.25%	20.03%	20.14%
	σ	7.39	9.33	7.02	7.83	6.36
H3	μ	18.79%	18.01%	26.22%	25.26%	11.73%
	σ	12.03	13.32	18.22	5.79	4.83

Table IV compares the results achieved by many neural network architectures [7]-[8] and our HCGA approach for the same data sets. The standard deviation (σ) refers to the mean, which was not given for the tests reported in [8]. $HCGA_2$ performs better than most other approaches in most of the cases, showing its great generalization capabilities when applied to classification problems. For instance, the $HCGA_2$ improvement over the second best ANN for the Card3 problem reaches nearly 20%, taking into account the classification error rate. Another advantage over most other ANNs is the

easiness of use, since no parameter adjustments have to be done (we used the default GA parameters for all tests).

TABLE IV

COMPARED RESULTS BETWEEN $HCGA_2$ AND OTHER ANNs.

Correctly classified patterns									
	card1			card2			card3		
	best	μ	σ	best	μ	σ	best	μ	σ
MLP	86.0	86.0	1.03	81.0	81.0	0.86	81.0	81.0	1.19
RBF	88.0	87.0	0.67	82.0	81.0	0.70	83.0	83.0	0.70
CasCor	84.0	78.0	2.48	79.0	77.0	1.87	82.0	80.0	1.97
Tower	84.8	—	2.41	78.4	—	2.91	79.6	—	2.26
Pyramid	86.0	—	1.87	79.0	—	2.97	79.0	—	3.45
DistAl	84.4	—	2.32	84.1	—	4.10	82.9	—	3.54
Upstart	90.7	—	2.03	86.1	—	1.96	83.7	—	2.53
PercCasc	87.6	—	2.12	86.8	—	2.97	84.1	—	2.11
$HCGA_2$	**87.2**	**86.4**	**0.97**	**84.9**	**83.4**	**1.68**	**86.6**	**84.2**	**1.61**
	heart1			heart2			heart3		
	best	μ	σ	best	μ	σ	best	μ	σ
MLP	80.0	80.0	0.96	82.0	82.0	1.14	76.0	76.0	1.12
RBF	82.0	82.0	1.20	82.0	81.0	1.72	79.0	79.0	0.48
CasCor	82.0	79.0	1.96	80.0	77.0	1.66	74.0	72.0	1.97
Tower	78.7	—	2.73	77.8	—	3.11	70.8	—	3.36
Pyramid	79.5	—	2.93	78.2	—	2.69	76.0	—	2.38
DistAl	79.1	—	4.03	80.9	—	2.92	82.6	—	2.71
Upstart	84.1	—	2.48	81.5	—	2.06	80.8	—	2.65
PercCasc	82.7	—	2.64	82.6	—	3.03	81.0	—	2.02
$HCGA_2$	**80.4**	**79.0**	**1.36**	**82.2**	**80.8**	**1.35**	**79.1**	**75.7**	**2.44**

D. Comparison with $HCGA_1$

The first version of this algorithm, developed by Coelho et al. [1], showed good results on classification problems, as well as on prediction problems, but had some problems that we have solved on this version.

First of all, the HNNs now have the possibility of augmenting and prunning [12] the number of neurons at the hidden layer at each generation, giving space to the arousal of "extend-shrink" behaviors. Another advantage of the new $HCGA_2$ is the separation of HNN's with different architectures into different GAs, so that the neurons from the NeuronGAs will always work with the same peers, facilitating the neuron's specialization. The creation of one NeuronGA for each *position* in the net's hidden layer, instead of one NeuronGA for each neuron AF type, as in $HCGA_1$, also brings advantages. In this way, even neurons with the same AF are encouraged to specialize into different regions of the search space (due to coevolutionary forces). What happened in $HCGA_1$ was that a HNN containing neurons with the same AF would have them to perform almost the same task, since they were part of the same NeuronGA.

Table V shows a comparison of the RMSE between the best HNN of $HCGA_1$ and $HCGA_2$. The difference to the results in Table IV is that, in $HCGA_1$, the validation patterns were added to the training patterns, the test patterns were used for validation and no independent test was made. Thus, to be able to compare both versions, we had to run on the same conditions. It is clear that $HCGA_2$ has much better results, surpassing the first version by an average of 45% for the card problem and by 20% for the heart problem.

TABLE V

COMPARED RESULTS BETWEEN $HCGA_1$ AND $HCGA_2$

Best network's root mean squared error						
	card1	card2	card3	heart1	heart2	heart3
$HCGA_1$	0.0988	0.1205	0.1105	0.1174	0.1435	0.1870
$HCGA_2$	**0.0349**	**0.0814**	**0.0640**	**0.1087**	**0.1044**	**0.1435**

E. Future Work

Some improvements could be undertaken as future work. First of all we should test the performance of the $HCGA_2$ on other kinds of problems, like temporal series prediction, function interpolation, etc. Likewise, we can extend the algorithm by using configurable *splines* and/or *polinomials* as AF's of the hidden layer neurons, instead of setting a predefined number of AF's. It would be also possible to use an ANN as AF, creating a nested structure for the HNN. Still we could add more levels to the $HCGA_2$ hierarchy, in order to co-evolve two or more neural networks that work together on a certain upper level problem.

IV. CONCLUSION

This work has examined the suitability of merging together in a same framework various promising approaches proposed recently for the configuration of neural net architectures, particularly those involving coevolution. The resulting model ($HCGA_2$) comprehends a new hierarchical-based evolutionary scheme devoted to the progressive assembling of heterogeneous neural structures. This blending strategy was assessed through a series of benchmarking tests over classification problems. The findings obtained so far corroborate other results already presented in the literature, showing that $HCGA_2$ constitutes a promising design strategy on the direction of fully automatic adjustment of an extended set of neural networks parameters.

A. Acknowledgments

This research was partially sponsored by **CAPES** and **FAPESP** (00/07631-6), through MSc. scholarships to the first and second authors respectively. The third and fourth authors would like to thank **CNPq** for their support (300910/96-7). The computational resources necessary to run the experiments were provided by **CENAPAD** (National Center of High Performance Computation in São Paulo), a project from UNICAMP / FINEP - MCT.

V. REFERENCES

[1] A. L. V. Coelho, D. Weingaertner and F. J. Von Zuben, "Evolving Heterogeneous Neural Networks for Classification Problems", *Procs. of Genetic and Evolutionary Computation Conference (GECCO-2001)*, pp.266-273, Morgan Kaufmann Publishers, July 2001.

[2] B. Whitehead and T. Choate, "Cooperative-Competitive Genetic Evolution of Radial Basis Function Centers and Widths for Time Series Prediction", *IEEE Trans. on Neural Networks*, Vol. 7:4, pp. 869-880, 1996.

[3] D. E. Moriarty, and R. Miikkulainen, "Efficient Reinforcement Learning through Symbiotic Evolution". *Machine Learning*, Vol. 22, pp.11-33. Kluwer Academic Publishers, Boston, 1996.

[4] D. E. Moriarty, and R. Miikkulainen, "Hierarchical Evolution of Neural Networks". *Proceedings of the IEEE International Conference on Evolutionary Computation*, pp.428-433, 1998.

[5] D. Whitley, "Genetic Algorithms and Neural Networks", In: Periaux, J. and Winter, G. eds., *Genetic Algorithms in Engineering and Computer Science*. John Wiley & Sons Ltd., 1995.

[6] E. Iyoda and F. Von Zuben, "Evolutionary Hybrid Composition of Activation Functions in Feedforward Neural Networks", *Procs. IJCNN*, article #396, 1999.

[7] J. Ribeiro and G. Vasconcelos, "An Experimental Evaluation of the Cascade-Correlation Network in Pattern Recognition Problems", *Proc. of ICONIP*, pp.1133-1136, Springer-Verlag, New Zeland, 1997.

[8] J. Ribeiro and G. Vasconcelos, "Constructive Neural Networks for Pattern Classification and Verification", *Proc. of ICONIP*, Springer-Verlag, 1999.

[9] L. Prechelt, "PROBEN1: A Set of Neural Benchmarking Rules", TR 21/94, Universtät Karlsruhe, 1994.

[10] Q. Zhao, "A Coevolutionary Algorithm for Neural Net Learning", *Procs. of ICNN*, Vol.1, pp.432-437, 1997.

[11] R. Parekh, J. Yang and V. Honavar, "Constructive Neural Network Learning Algorithms for Multi-Category Real-Valued Pattern Classification",TR 97-06, Dep. of Computer Science, Iowa State University, 1998.

[12] R. Reed, "Pruning Algorithms – A Survey", *IEEE Trans. on Neural Networks* 4:5, pp.740-747, 1993.

[13] S. S. Haykin, "Neural Networks: A Comprehensive Foundation", Prentice Hall, 1998.

[14] T. Bäck, D. B. Fogel and T. Michalewicz, editors, "Evolutionary Computation 1: Basic Algorithms and Operators", Institute of Physics Publishing, 2000.

[15] X. Yao, "A review of evolutionary artificial neural networks", *Int. J. Intell. Syst.*, Vol. 8:4, pp. 539-567, 1993.

[16] Y. Liu and X. Yao, "Evolutionary Design of Artificial Neural Networks with Different Nodes", *Procs. of the Third IEEE International Conf. on Evolutionary Computation (CEC96)*, pp. 670-675, Japan, May 1996.

[17] Z. Michalewicz, G. Nazhiyath and M. Michalewicz, "A note on the usefulness of geometrical crossover for numerical optimization problems", *Proc. 5th Ann. Conf. on Evolutionary Programming*, MIT Press, 1996.

VI. APPENDIX A (HCGA$_2$ ALGORITHM)

1) Generate an initial population of NNdGA_POPULATION_SIZE NNds for the NNdGA. These NNds contain the number of neurons in the HNN's hidden layer (ranging from MIN_HIDDEN_NEURONS to MAX_HIDDEN_NEURONS) and the type of each neuron's AF.

2) For each new NNd, create a genetic algorithm (HNNGA) that will be used to train the HNNs.

- **a.** Create one GA (NeuronGA) for each neuron on the NNd, and initialize it with a population of NeuronGA_POPULATION_SIZE neurons, which have all the same AF (determined by the NNd).
- **b.** Generate HNNGA_POPULATION_SIZE initial HNNs taking the neurons of their hidden layer from the NeuronGAs. Those neurons will be taken randomly, one from each NeuronGA, in order to create networks hat are equivalent to the NNd. One neuron may be used by more than one HNN.

3) Run each HNNGA for HNNGA_MAX_GENERATIONS.

- **a.** Apply mutation (HNNGA_MUTATION_CHANCE) to the HNNGA population. Add to the population: 1) HNNGA_CROSSOVER_CHANCE HNNs generated by two points crossover; 2) HNNGA_RANDOM_PERCENT randomly generated HNNs.
- **b.** Adjust the weights of the output layer neurons of each HNN using the LMS method.
- **c.** Use the validation input/output patterns to compute the Root Mean Squared Error (RMSE) of each HNN and set their fitness as being (1.0 – RMSE).
- **d.** Send the HNNs' fitness to the neurons in their hidden layer. The fitness of the neurons will be the average fitness of all networks they participate in.
- **e.** HNNGA_ELITIST_SELECTION best HNN's will pass to the next generation. The population will be completed by tournament selection between the remaining HNN's.
- **f.** Run NeuronGA_MAX_GENERATIONS evolutionary cycles for each NeuronGA.
  - **i)** Set fitness on unused neurons. They receive as fitness the average fitness of the NeuronGA, divided by the number of cycles they are unused. After NeuronGA_MAX_UNUSED_CYCLES the fitness is set to zero.
  - **ii)** Generate a new neuron population by taking: 1) the neurons that belong to the best ranked HNNs (regardless of their fitness); 2) the NeuronGA_ELITIST_SELECTION best neurons; 3) NeuronGA_CROSSOVER_CHANCE neurons generated by geometrical crossover [17]; 4) NeuronGA_RANDOM_PERCENT randomly generated neurons; and complete the population with tournament selection between the remaining neurons. Finally, apply uniform mutation [14] (NeuronGA_MUTATION_CHANCE) to the population.

4) Take as fitness for each NNd the fitness of the best HNN evolved by its HNNGA.

5) If NNdGA has run for NNdGA_MAX_GENERATIONS, go to **9)**.

6) Apply mutation (NNdGA_MUTATION_CHANCE) to the NNds. Add to the population: 1) NNdGA_CROSSOVER_CHANCE NNds generated by two points crossover; 2) NNdGA_RANDOM_PERCENT randomly generated HNNs.

- **a.** Crossover is done between the best ranked NNds. The NeuronGAs corresponding to the exchanged neurons also have to be exchanged and all HNNs of the HNNGAs have to be updated, in order to reflect the changes in the NNds.
- **b.** Mutation can add or remove a neuron, as well as change the type of its AF. Again, the NeuronGA's and the HNN's in the HNNGA's have to be adapted to reflect the changes.

7) The NNdGA_ELITIST_SELECTION best NNd's will pass to the new generation. The population will be completed by tournament selection between the remaining NNd's.

8) Go to step **2)**.

9) Return the best ranked HNN of the NNd with highest fitness and **exit**.