Optimization of Neural Networks Weights and Architecture: A multimodal methodology

Antonio Miguel F. Zarth and Teresa B. Ludermir
Federal University of Pernambuco
Center of Informatics
Av. Luis Freire s/n CEP 50740-540, Cidade Universitária, Recife-PE, Brazil
amfz@cin.ufpe.br, tbl@cin.ufpe.br

***Abstract*—This paper describes a multimodal methodology for evolutionary optimization of neural networks. In this approach, we use Differential Evolution with parallel subpopulations to simultaneously train a neural network and find an efficient architecture. The results in three classification problems have shown that the neural network resulting from this method has low complexity and high capability of generalization when compared with other methods found in literature. Furthermore, two regularization techniques, weight decay and weight elimination, are investigated and results are presented.**

I. Introduction

The use of evolutionary-based algorithms to optimize artificial neural networks (ANNs) has been a popular approach when addressing the shortcomings of a tedious process of trial and error while looking for efficient network architecture and an appropriate training algorithm. In an empirical setting, the eventual success of an ANN project greatly depends on professional experience, which guides the appropriate selection of network architecture, training algorithm, learning rates, etc. Both training and architecture design can be considered optimization problems, so the use of stochastic optimization methods has been found to be a promising alternative for optimizing ANNs.

Evolutionary approaches for optimizing ANNs can be classified into two major types [1]: noninvasive and invasive. The first one refers to approaches where evolutionary selection is used but some gradient training is required for fitness evaluation. On the other hand, an invasive approach tries to optimize weights and architecture in the evolution process. The latter, which refers to invasive approaches, is better for the generation of efficient networks, avoiding gradient-based fitness evaluation, resulting in a more robust search coverage [1].

Nevertheless, optimization of ANNs weights and architecture by stochastic methods may lead to an efficient ANN in an extended convergence speed. According to our review of literature, two typical methodologies which concern simultaneous optimization of weights and architectures involve constructive and pruning methods [2] [3] [1]. Constructive methods tend to build small networks due to their incremental learning nature while pruning methods need to know a priori how large the original network should be [4]. However, some constructive approaches may need to reinitialize the entire population when the architecture grows up, discarding valuable search information from the previous search, and demanding more time to converge.

This paper presents a methodology for the simultaneous optimization of feedforward network weights and selection of an appropriate architecture, in a constructive way, using differential evolution. The main contribution of the proposed methodology focuses on developing a systematic procedure for optimizing weights, attempting to maintain the diversity while simultaneously evaluating and exchanging different architectures in multiple populations. The implicit multimodal measure, called island model [5], is used to grow the network architecture when necessary while the operators of differential evolution optimize weights. As an attempt to improve the generalization performance, 2 techniques of regularization are investigated (weight decay and weight elimination), and results are compared.

In experiments to validate the method, simulations in three well-known classification problems are performed: (1) Breast Cancer data set; (2) Diabetes diagnosis in Pima Indians and (3) Thyroid dysfunction data set.

The next section describes the differential evolution training algorithm. The proposed methodology is present in Section 3. The experiments performed and discussions are presented in Section 4. Section 5 contains the final remarks.

II. Differential Evolution Training Algorithm

Differential Evolution (DE) is a population-based and direct search method proposed by Storn and Price [6] in 1995. It is a heuristic algorithm for global optimization, and operates with decision variables in real number form. As such, it can be applied to global searches within the weight space of a typical feed-forward neural network [7] [8]. In standard training processes, both the input vector X and the output vector Y are known, and the synaptic weights in W are adapted to obtain appropriate functional mappings from the input X to the output Y. The optimization goal is to minimize the objective function by optimizing the values of the network weights.

Differential evolution resembles the structure of an evolutionary algorithm (EA) and starts with a randomly generated initial population vector. In this study we used the classical DE (DE/rand/1/BIN). The following process will be executed as long as the termination condition is not fulfilled: for each individual in the population, an offspring is created by adding the weighted difference of two randomly selected vectors to a third randomly selected vector. Through the computation of the distance between two randomly selected individuals, DE determines a function gradient in a given area, instead of in a single point. The main operators of DE are:

A. *Mutation.*

Let us assume that $x_{ri,G}(i = 1, 2, ...N_p)$ are solution vectors in generation G, where N_p is the population size. On optimization of neural networks, each position of the vector represents a weight or bias of the neural network. For each $x_{ri,G}$, a mutant vector is defined by

$$V_{ri,G} = X_{r1,G} + F \times (X_{r2,G} - X_{r3,G}) \qquad (1)$$

where $i = 1, 2, , N_p$ and $r1, r2, r3$, are mutually different, randomly selected indices. F is the mutation factor, which provides the amplification to the difference between two individuals $X_{r2,G} - X_{r3,G}$ so as to avoid search stagnation and it is usually taken in the range of [0, 1].

B. *Crossover.*

DE utilizes the crossover operation in order to generate new solutions and also to increase the diversity of the population. A vector of solutions is randomly selected from the mutant individuals when a random number is less than a crossover constant $C_r \in [0, 1]$.

$$U_{ji,G} = \begin{cases} V_{ji,G} & if\ rand_j(0,1) < C_r \vee j = k \\ X_{ji,G} & \end{cases} \qquad (2)$$

$j = 1, 2, ..., D$, where D is the dimension of problem.

DE's selection scheme also differs from other evolutionary algorithms. If the trial vector $U_{ji,G}$ has a better fitness value than that of its target vector, $U_{ji,G}$ replaces the target vector in the next generation. Assuming that the objective function is to be minimized, the vector with the lower objective function value wins a place in the next generations population.

III. Optimization Methodology

Recently, several studies investigated the use of DE in the training of neural networks [7] [8]. One of the most detailed is the study of Ilonen et al., 2003 [7], which concludes that the results of training ANNs with DE did not reveal distinct advantage in time or quality of solution compared to methods of training by gradient, but convergence to a global minimum can be expected. However, as the author concludes, it is clear that the use of DE can be successfully applied in different cases:

- When the network is very large;
- When the problem often shows local minimum;
- When there is much noise in the data.

Another valuable information, but seldom explored by [7], is the influence of DE operators in the training of neural networks. The weights of RNA are very sensitive to sudden changes, which occur when the amplification parameter F is very high. Otherwise, lower values cause faster convergence, but the diversity of the population falls rapidly, undermining the evolutionary process. To avoid premature convergence, a multimodal approach can be used to explore different regions of the search space.

DE, like other EAs, is easily parallelized due to the fact that each member of the population is individually evaluated. Moreover, this implicit measure, the so called island model [5], becomes attractive for high-performance scientific computing. This acceptance has been facilitated by two major developments: massive parallel processors and the widespread use of distributed computing [9]. Furthermore, in this approach we explore how Differential Evolution can train ANNs and select an appropriate architecture using parallel populations.

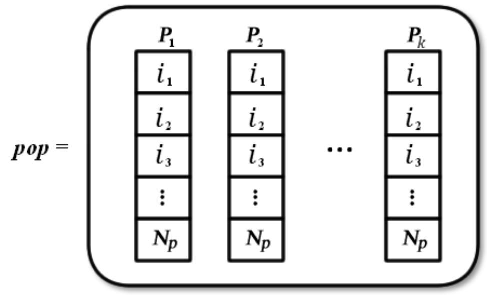

Figure 1. Representation of the population

A. *Representation.*

In this paper, all ANNs topologies have a single hidden layer, containing only connections between adjacent layers. The ANN topology contains N_1 input nodes, N_2 hidden nodes, and N_3 output nodes. N_1 and N_3 are problem-dependent, according to data preprocessing and to the number of input features and outputs.

Assume that there exists N_p solution vectors in each K subpopulation, as shown in Fig. 1. The proposed model specifies, for each subpopulation, a population of ANNs whose architectures differ from others. The subpopulation P_1 is initially formed by N_p neural networks with minimal architecture, i.e. only 1 neuron in the hidden layer. The subpopulation P_2 will have a population with 2 neurons in the hidden layer and so on.

Each solution i is composed by the vector W containing real numbers which represent the network weights w:

$$W \equiv (W_1, W_2). \quad (3)$$

$$W_1 \equiv (w_1, w_2, ..., w_{N_{max1,s}}), w_j \in \Re, j = 1, 2, , N_{max1,s}. \quad (4)$$

$$W_2 \equiv (w_1, w_2, ..., w_{N_{max2,s}}), w_j \in \Re, j = 1, 2, , N_{max2,s}. \quad (5)$$

where $N_{max1,s} = N_1 \times N_{2,s}$ and $N_{max2,s} = N_{2,s} \times N_3$, and $N_{2,s}$ is the number of neurons in the hidden layer on the island s.

B. Migration.

Similar to all parallel population-based algorithms, each subpopulation evolves independently toward a solution. To promote information sharing, the best individual of each subpopulation is moved to other subpopulations, according to a predefined topology. This operation is called "migration" [9]. After a fixed number of epochs, a number of individuals are selected from each subpopulation to be exchanged with neighboring subpopulations.

When using subpopulations with individuals of different sizes, this process cannot be achieved directly. To address this problem, we propose the use of unidirectional migration, changing individuals of each subpopulation to the immediately subsequent one (with one more neuron), as shown in Fig. 2. Before migration, selected individuals will receive an inactive neuron in its architecture, i.e new connections set to zero:

$$W_{new} = (W_1, Z_1, W_2, Z_2). \quad (6)$$

$$Z_1 \equiv (w_1, w_2, ..., w_{N_1}), w_j \in \{0\}, j = 1, 2, ..., N_1. \quad (7)$$

$$Z_2 \equiv (w_1, w_2, ..., w_{N_3}), w_j \in \{0\}, j = 1, 2, ..., N_3. \quad (8)$$

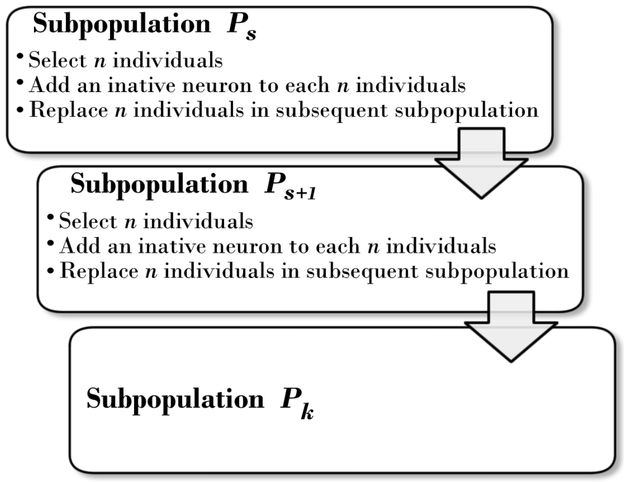
Subpopulation P_s
• Select n individuals
• Add an inative neuron to each n individuals
• Replace n individuals in subsequent subpopulation
Subpopulation P_{s+1}
• Select n individuals
• Add an inative neuron to each n individuals
• Replace n individuals in subsequent subpopulation
Subpopulation P_k

Figure 2. Unidirectional migration of $S \in 1, 2, ..., k$ subpopulations

In each subpopulation, n individuals are selected. A neuron is added with synaptic connections with zero (Eq: 6) and then migrates to the immediately subsequent subpopulation with higher architecture representation. This unusual procedure ensures the growth of the network without losing its ability to generalize. It is easy to see that the individuals of the last subpopulation (which has the largest representation of architecture) do not migrate, and that the subpopulation P_1 (with individuals with small architecture) will be the only one that does not receive individuals from others.

C. Network growth.

Through the representation of heterogeneous architectures, the evolutionary process can train and measure the performance of the architecture contained in subpopulations within range $[P_1, P_k]$. The best fitness of each subpopulation informs to the evolutionary process an estimate on the complexity of the network needed for the problem. The network growth, in addition to representation contained in individuals of the P_k, is explained as follows.

The evolutionary process can maintain the diversity of the population by exchanging individuals between subpopulations at intervals of pre-defined epochs. Thus, an estimate of the necessary architecture size for the convergence of the neural network is given according to the best individual of the best subpopulation. However, this pre-set approach limits the search for architectures in the range $[P_1, P_k]$, which may be insufficient to solve a particular problem.

After the convergence of the evolutionary process according to the stopping criterion, we know what are the best individual of the entire population and the best individual of each subpopulation. Under the context of this methodology, if the best individual of the subpopulation P_1 (with minimal network architecture) is the worst when compared to the best individuals from other subpopulations, probably this architecture is not good enough to generalize the problem. Thus, the population P_1 is excluded and a new subpopulation P_{K+1} is created. Thus, the first subpopulation P_1 now has two neurons in the hidden layer, P_2 will have three neurons and so on.

This process is repeated until the best individual of the subpopulation P_1 is not the worst if compared to the best individuals of all subpopulations. The pseudo code of this process is described in Algorithm 1.

D. Stopping criterion.

The stopping criterion is based on the proposal by [1]. To avoid overfitting, at the end of every ten-generation the stopping criterion is measured on the validation set. If there is overfitting in 20 consecutive intervals (200 generations) in each subpopulation, the training stops.

E. Cost Function

The qualitative measure during the optimization process is given by:

$$E(T) = CEP + MSE. \quad (9)$$

where T is the training set, MSE is the mean square error and CEP is the classification error percentage, defined by:

$$CEP = \frac{100}{\#T} \sum_{x \in T} \varepsilon(x). \tag{10}$$

where $\#T$ is the number of patterns in the set T and $\varepsilon(x)$ is the correctly classified training pattern.

Additionally, we combined the error measure with two well-known regularization techniques, in an attempt to improve the generalization. A bias term β is added to the original cost function $E(T)_0$:

$$E(T) = E(T)_0 + \mu\beta. \tag{11}$$

where μ is the parameter for the importance of the bias term β, defined as follows.

Weight Decay (WD) tries to decrease all weights uniformly adding a penalty term on the error function. The weight decay penalty term (Eq. 12) causes the weights to converge to smaller absolute values than they otherwise would. Excessive large weights may lead to the output function being too rough and can cause excessive variance of the output.

$$\beta = \frac{\|w\|^2}{\psi}. \tag{12}$$

where w is the vector of weights. Assuming we are evaluating different architectures simultaneously, we divide this penalty term to the number of connections of the network ψ, avoiding superior errors on larger architectures.

On the other hand, *Weight Elimination* (WE) [10] uses a cost function to reduce the number of synaptic connections. It is possible to decay small weights more rapidly than large weights [11]. According to [12], these small weights result in a poor generalization, given the high probability of assuming completely arbitrary values. The weight elimination is given by:

$$\beta = \sum_{i \in_{total}} \frac{(w_i/w_0)^2}{1 + (w_i/w_0)^2}. \tag{13}$$

where w_0 is a pre-defined parameter.

IV. Experiments and Results

In order to validate the hybrid methodology for optimization of neural networks, we used 3 well-known classification problems. Our results are compared with other studies that used the same methodology of experimentation. In this work, we also performed the experiments with 2 regularization techniques, WD and WE. The data sets are: breast cancer with 699 examples and 2 classes; thyroid with 7200 examples and 3 classes; diabetes with 768 examples and 2 classes, both available in [13].

For each experiment, the data sets were randomly partitioned with stratification, i.e., maintaining the same proportions of the number of classes for the training (50%), validation (25%) and test sets (25%). Exceptionally for the cancer data set, the 3 permutations suggested by [13] were used, in order to achieve similar comparisons to the results of this author and with the work of [1], using the same procedure.

Algorithm 1 Multimodal optimization pseudocode

```
1: $k \leftarrow$ Number of populations, with $k > 1$
2: $n \leftarrow$ Initial number of neurons in hidden layer
3: $N_p \leftarrow$ number of individuals in each subpopulation
4: $m \leftarrow$ migration step
5: repeat
6:     $n' \leftarrow n$
7:     $n \leftarrow n + 1$
8:     for $s = 1$ to $k$ do
9:         if $P_{s+1}$ not exists then
10:            Initialize population $P_s$ with $N_p$ individuals with $n'$ neurons
11:        else
12:            $P_s \leftarrow P_{s+1}$
13:        end if
14:        $n' \leftarrow n' + 1$
15:    end for
16:    while stopping criterion not reached do
17:        Perform a DE step at each population
18:        Evaluate all individuals of all subpopulations
19:        if $epoch$ is a multiple of $m$ then
20:            Perform unidirectional migration
21:        end if
22:    end while
23:    $worst \leftarrow 1$
24:    $best' \leftarrow$ fitness of fittest individual of subpopulation $P_1$
25:    for $s = 1$ to $k$ do
26:        $best_s \leftarrow$ fitness of fittest individual of subpopulation $P_s$
27:        if $best_s > best'$ then
28:            $worst \leftarrow s$
29:            $best' \leftarrow best_s$
30:        end if
31:    end for
32: until $worst \neq 1$
```

The neural network has only one hidden layer and all neurons have the tangent transfer function. The parameters used in the experiments were:

- Number of subpopulations: 3;
- Number of individuals on each subpopulation: 30;
- Step for migration: 50 seasons;
- Number of individuals who migrate: 5 best selected;
- F: 0.3 (Equation 1);
- Cr: 0.7 (Equation 2);

Experiments with the cancer datasets were performed 30 times each one. With diabetes and thyroid we performed $30*$

Table I
MEAN CEP AND THE NUMBER OF HIDDEN NEURONS FOR RESULTS OF THE CANCER DATA SET, USING THE 3 PERMUTATIONS SUGGESTED BY [13].

text		L-BP	P-BP	N-BP	MGNN	DE Multimodal
Cancer 1	Class.(%)	2.93	1.47	1.38	3.14	**1.11**
	σ	0.18	0.60	0.49	0.01	0.68
Cancer 2	Class.(%)	5.00	4.52	4.77	6.41	**4.50**
	σ	0.61	0.70	0.94	0.01	0.80
Cancer 3	Class.(%)	5.17	3.37	3.70	4.61	**3.31**
	σ	0.00	0.71	0.52	0.01	0.83

10 cross-validation. In all experiments the results refers to the average classification error percentage (CEP) on the test set. Statistical t-test were performed to evaluate the results.

Table I shows the results compared with the invasive technique MGNN proposed by [1] and with 3 types of architectures investigated by [13]: L-BP (Linear Backpropagation), P-BP (Pivot Backpropagation) , N-BP (No shortcut Backpropagation). While MGNN is more reliable, with standard deviation of 0.1 in all the permutations, the proposed approach achieved significantly lower errors. In the best result on the proposed methodology, the permutation cancer1 obtained classification error of 1.11%, better result according t-test with 95% confidence interval. In permutation cancer2 we obtained a mean error of 4.50% and 3.31% in permutation cancer3. Both statistically similar to the best results found in [13] experiments according to the t-test with 95% confidence level. In all cancer experiments we achieved lower errors and smaller networks than [1].

Table II
MEAN CEP AND THE NUMBER OF HIDDEN NEURONS FOR RESULTS OF DIABETES AND THYROID.

		Ludermir et al, 2006 [14]	Multimodal Methodology
Diabetes	Class.(%)	25.87	24.14
	hidden	4.53	3.23
Thyroid	Class.(%)	7.32	3.08
	hidden	7.05	5.6

The results expressed in Table II refer to results in 2 problems of classification partitioned randomly with stratification. The results are compared with the noninvasive technique of [14], that combines *tabu search*, *simulated annealing* and *backpropagation* for both training and adjusting the architecture of an neural network. The proposed method achieved lower errors in both data sets, with 3.08% error in the thyroid data set and 24:14% in the diabetes data set. The resulting architecture was also smaller, reinforcing the hypothesis that constructive methods tend to result in smaller networks. The proposed method obtained a solution with an average of 3.23 neurons in hidden layer on the diabetes data set and 5.6 neurons on the thyroid data set.

The use of weight elimination (WE) in the proposed optimization methodology did not present interesting results in any experiments, as well as weight decay (WD) also fail in thyroid and diabetes dataset, as shown on table III. Perhaps better results can be achieved by selecting an appropriate trade-off between generalization error and complexity of architecture.

Table III
MEAN (CEP) AND STANDARD DESVIATION (σ) FOR THE METHODOLOGY PROPOSED WITH AND WITHOUT REGULARIZATION.

		DE	DE:WD	DE:WE
Cancer 1	Class.(%)	1.57	1.11	1.95
	σ	0.66	0.68	0.66
Cancer 2	Class.(%)	4.50	4.51	4.85
	σ	0.80	0.71	1.02
Cancer 3	Class.(%)	3.63	3.31	3.78
	σ	0.98	0.83	0.87
Diabetes	Class.(%)	24.14	24.31	24.33
	σ	2.37	2.54	2.32
Thyroid	Class.(%)	3.08	6.77	6.65
	σ	1.76	1.57	1.79

V. FINAL REMARKS

This paper presented a multimodal methodology for optimizing neural networks using differential evolution. The results indicate that the evolutionary optimization of weights and architecture of neural networks is a viable alternative considering the limitations of most traditional methods. However, this approach does not perform exhaustive search, i.e., does not have to restart the training as the network grows, losing previous search information. The results indicate that constructive methods tend to get smaller networks than pruning methods, without loss of generalization quality.

Most of the current structural evolutions are governed by static rules that are refined to a particular problem but may be inefficient to another one [1]. The proposed model allows a parallel search of architectures based on estimation of the best individuals of each subpopulation while improving the diversity of the population.

Future investigations should consider the search for any multilayer architecture. The current model uses only one hidden layer. Furthermore, we intend to review an efficient adaptable parameter for WE and WD cost function, allowing an efficient trade-off between residual error and model complexity.

References

[1] P. Palmes, T. Hayasaka, and S. Usui, “Mutation-based genetic neural network,” *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 587–600, 2005.

[2] X. Yao, “Evolving artificial neural networks,” *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.

[3] M. Carvalho and T. Ludermir, “Particle swarm optimization of neural network architectures and weights,” in *7th International Conf. on Hybrid Intelligent Systems, 2007.*, 2007, pp. 336–339.

[4] L. Ma and K. Khorasani, “New training strategies for constructive neural networks with application to regression problems,” *Neural Networks*, vol. 17, no. 4, pp. 589 – 609, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/B6T08-4C0TCD2-1/2/4d14f15099df2acaf67d1150c8d516a0

[5] A. Eiben and J. Smith, *Introduction to evolutionary computing*. Springer, 2003.

[6] R. Storn and K. Price, “Differential evolution – a simple efficient adaptive scheme for global optimization over continuous spaces,” Int. Compt. Sci. Inst., Berkeley, CA, Tech. Rep. 95-012, 1995.

[7] J. Ilonen, J.-K. Kamarainen, and J. Lampinen, “Differential evolution training algorithm for feed-forward neural networks,” *Neural Processing Letters*, vol. 17, no. 1, pp. 93–105, 2003.

[8] A. Slowik and M. Bialko, “Training of artificial neural networks using differential evolution algorithm,” in *Conf. on Human System Interactions, 2008*, 2008, pp. 60–65.

[9] D. Tasoulis, N. Pavlidis, V. Plagianakos, and M. Vrahatis, “Parallel differential evolution,” in *Congress on Evolutionary Computation, 2004*, vol. 2, 2004, pp. 2023–2029.

[10] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman, “Generalization by weight-elimination with application to forecasting,” in *Advances in neural information processing systems 3*, 1990, pp. 875–882.

[11] C. Zanchettin and T. B. Ludermir, “Global optimization methods for designing and training feedforward artificial neural networks,” *Dynamics of Continuous, Discrete and Impulsive System*, vol. 14, pp. 328–337, 2007.

[12] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, 1998.

[13] L. Prechelt, “PROBEN1 — A set of benchmarks and benchmarking rules for neural network training algorithms,” Fakultät für Informatik, Universität Karlsruhe, Tech. Rep. 21/94, 1994, anonymous FTP: /pub/papers/techreports/1994/1994-21.ps.Z on ftp.ira.uka.de.

[14] T. Ludermir, A. Yamazaki, and C. Zanchettin, “An optimization methodology for neural network weights and architectures,” *IEEE Transactions on Neural Networks*, vol. 17, no. 6, pp. 1452–1459, 2006.