Use of Backpropagation and Differential Evolution algorithms to training MLPs

Luiz Carlos Camargo, Hegler Correa Tissot, Aurora Trinidad Ramirez Pozo
Departamento de Informática (INF)
Universidade Federal do Paraná (UFPR)
Curitiba, Paraná, Brasil
{lccamargo, hctissot, aurora}@inf.ufpr.br

Abstract. **Artificial Neural Networks (ANNs) are often used (trained) to find a general solution in problems where a pattern needs to be extracted, such as data classification. Feedforward (FFNN) is one of the ANN architectures and multilayer perceptron (MLP) is a type of FFNN. Based on gradient descent, backpropagation (BP) is one of the most used algorithms for MLP training. Evolutionary algorithms can be also used to train MLPs, including Differential Evolution (DE) algorithm. In this paper, BP and DE are used to train MLPs and they are both compared in four different approaches: (a) backpropagation, (b) DE with fixed parameter values, (c) DE with adaptive parameter values and (d) a hybrid alternative using both DE+BP algorithms.**

Artificial Neural Network; Multilayer Perceptron; Backpropagation (BP) algorithm; Differential Evolution (DE) algorithm.

I. INTRODUCTION

Artificial Neural Networks (ANNs) are often used to detect trends that are too complex to be perceived by humans. ANNs are specially used to find a general solution to in problems where a pattern needs to be extracted, such as data classification. The main difficulty to apply ANN in some domain problem is to train the ANN to learn and predict. ANN provides different ways to solve many nonlinear problems that are hard to solve by conventional techniques. Good solutions for these problems can be found, but they depend on the training strategy adopted. Many algorithms are applied for the minimization and improvement of ANN learning. Theses algorithms can be classified into local minimization and global minimization. Local minimization algorithms, such gradient descent, are fast, but converge for local minima. On the other hand, global minimization algorithms use heuristic strategies to escape local minima [1].

Evolutionary algorithms [2] can help to avoid the problem of convergence to local minima and exploit global minimization techniques. Some evolutionary algorithms which can use such global strategy on ANN training are: the differential evolution (DE) algorithm [3], [5]; particle swarm optimization (PSO) algorithm [6]; ant colony optimization (ACO) algorithm [11]; and artificial bee colony (ABC) optimization [12].

Regardless of the algorithm used to train ANN, attention to weight values initialization and adjustment is fundamental for convergence and to avoid overfitting, a problem that usually happens in training patterns [I]. Specialized learning algorithms are used to adjust the weight values [5], and one of the most popular is error backpropagation (BP) [13], [14]. BP, based on gradient method, is efficient to train ANNs to solve hard problems. However, this algorithm is sensitive on local minima and requires a learning coefficient adjustment. When the chosen learning coefficient is too small or too high, it causes oscillations on the algorithm. Therefore, satisfactory results require a huge number of iterations [5].

As an alternative, evolutionary algorithms can be applied to search the weight of a FFNN escaping from local minima [3], [9].

Regarding the use of other evolutionary algorithms, the use of differential evolution (DE) algorithm was motivated by the chance of verifying and validating some DE inherent characteristics, such as its ability to reach a global minima of objective function, quick convergence, and a small number of parameters to be set up still with the possibility to adapt some of these parameters during the execution.

This paper describes and compares the results obtained in ANN training with different algorithms, datasets and parameters. To support this experiment, two algorithms in four different approaches were used: (a) backpropagation algorithm, (b) DE algorithm with fixed parameter values, (c) DE algorithm with adaptive parameter values and (d) a hybrid alternative using both algorithms.

This article is organized as follows: Section II is a brief presentation of feedforward neural network (FFNN) and multilayer perceptron (MLP); in Section III, Differential Evolution algorithm and its pseudocode are explained; Section IV describes Backpropagation algorithm in ANN training process; Section V, some related work, in Section VI, experiment description and results are shown; in Section VII concludes the paper.

II. FEEDFORWARD NEURAL NETWORKS (FFNN)

A Feedforward Neural Network (FFNN) is formed by interconnecting process units known as neuron (or nodes), and has the natural tendency to store experiential knowledge and to make it available for use [4]. Thus, they can exhibit basic characteristics of human behaviour such as: learning, association and generalization.

Basically, there are two kinds of FFNN: single-layer perceptron (SLP), and multilayer perceptron (MLP). The SLP networks consist of a single layer of output nodes, which are fed directly by input layer via a set of weights. MLP networks consist of multiple layers: an input layer, one or more hidden layers and an output layer. Each layer has nodes and each node is fully weighted interconnected to all nodes in the subsequent layer. The MLP transforms inputs to outputs through of nonlinear function as:

$$x_o = f\left(\sum_{h=1}^{H}(x_h * w_{h,o})\right) \tag{1}$$

where f() is the activation function of the *o*th output neuron, x_h is the output of *h*th hidden layer neuron and $w_{h,o}$ is the interconnection between *h*th hidden layer neuron and *o*th output layer neuron. H is the hidden layer size. Normally, the sigmoid function is most used as the activation function; it is given as follows [15]:

$$x_o = \frac{1}{1 + e^{\left(-\sum_{h=1}^{H}(x_h * w_{h,o})\right)}} \tag{2}$$

The activation function is applied in the *h*th hidden layer node. In the same way, it is also applied in the *o*th output layer node. Based on the differences between calculated output and the target value an error is defined as follows [15]:

$$E = \frac{1}{N} * \sum_{s=1}^{N}\left(\frac{1}{2} * \sum_{o=1}^{L}\left(t_o^{(s)} - x_o^{(s)}\right)^2\right) \tag{3}$$

where N is the number of patterns in data set, L is the number of output nodes, $t_o^{(s)}$ is the desired output value in the *o*th output neuron for the *s*th sample in data set, $x_o^{(s)}$ is the calculated output value in the *o*th output neuron for the *s*th sample in data set. The aim is to reduce the error E by adjusting interconnections (weights) between layers. Input and hidden layers can be set with an additional node called bias. Bias is a neuron in which its activation function is permanently set to 1. Just like other neurons, the bias is also weighted connected to the neurons in the subsequent layer. In Fig.1, a MLP model used in this paper is shown.

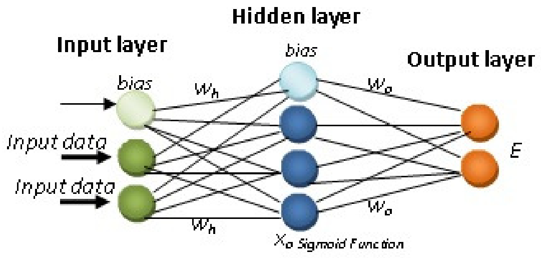


Figure 1. MLP Model

According to Fig.1, bias receives a value equals to 1; w_h are weights for all connections between input layer and the hidden layer; w_o are weights for all connections between hidden layer and the output layer; x_o is the sigmoid function used as activation function for hidden and output layers; finally, E is a function that computes MLP error, comparing output layer result with target values.

III. Differential Evolution (DE) Algorithm

Developed in 1997 by Kenneth Price and Rainer Storm [5], DE algorithm has been successfully applied for solving complex problems in engineering, reaching very close optimum solutions. DE can determine the size of mutation largely based on the current variance in the population, because it has mechanisms for adaptive mutation. For each member *i* a new child is generate by picking three individuals from the population and performing some vector additions and subtractions among them. The idea is to mutate away from one of the three individuals ($\vec{a}$) by adding a vector to it. This vector is created from the difference between the other two individuals ($\vec{b}$) – ($\vec{c}$). If the population is spread out ($\vec{b}$) and ($\vec{c}$), are likely to be far from one another and this mutation vector is large, else it is small. If the population is spread throughout the space, mutations will be much bigger than when the algorithm has later converged on fit regions of the space. The child is then crossed over with $\vec{i}$ [2].

DE is described as a stochastic parallel search method, which utilizes concepts borrowed from the broad class of evolutionary algorithms (EAs) [6]. DE, like other EAs, is easily parallelized due to the fact that each member of the population is evaluated individually [6]. DE is good and effective in nonlinear constraint optimization and is also useful for multimodal problems optimization [7]. The advantages over traditional genetic algorithm are: easy to use; efficient memory utilization, lower computational complexity and lower computational effort [7].

Moreover, DE meets fully the requirements [2] [7]:

a) Capacity to handle non-differentiable, nonlinear and multimodal cost functions;

b) Parallelizability to cover computation intensive cost functions;

c) Few control variables to manage the minimization, which are robust and easy to choose;

d) Compatible convergence to the global minimum.

The Figure 2 shows a typical implementation of DE, which represents a population as a vector and not a collection. DE always uses vector representations for individuals.

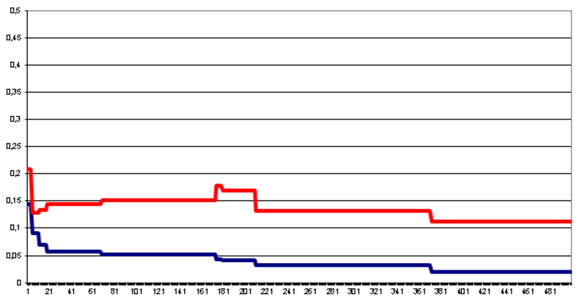
```
1: F ← mutation rate
2: CR ← crossover rate
3: P ← {} (Empty population of length popsize)
4: Q ← {} (Empty population of length popsize)
5: for i from 1 to popsize do
6:    Pi ← new random individual
7: Best ← {}
8: repeat
9:    for each individual Pi ∈ P do
10:       AssessFitness(Pi)
11:       if Q ≠ {} and Fitness(Qi) < Fitness(Pi)
then
12:          Pi ← Qi
13:       if Best={} or Fitness(Pi) <
             Fitness(Best) then
14:          Best ← Pi
15:    Q ← P
16:    for each individual Qi ∈ Q do
17:       a ← Copy (Qrand1) (Qrand1 ≠ Qi)
18:       b ← Copy (Qrand2) (Qrand2 ≠ Qi,a)
19:       c ← Copy (Qrand3) (Qrand3 ≠ Qi,a,b)
20:       d ← a + F (b - c)
21:    Pi ← one child from
             Crossover(d,Copy(Qi))(CR)
22: until Best is the ideal solution or timeout
23: return Best
```

Figure 2. DE Algorithm, adapted from [2]

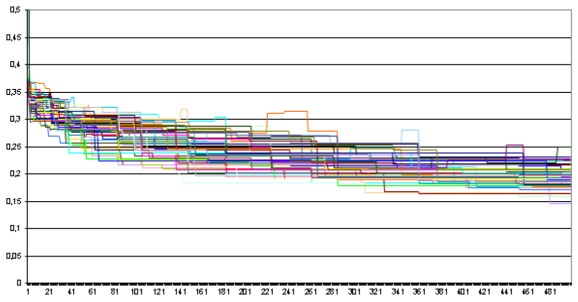
According to pseudocode, after creating initial population (P), element fitness must be calculated (line 10) and compared one to others to choose the best element in the population (lines 13-14). For each element in population a child d is created combining three other elements randomly chosen (lines 17-20). Crossover is applied between child and its parent and one of the result elements Pi is chosen (line 21). Chosen elements are then compared to its parents and can replace them when children fitness is better than their parents (lines 11-12). F is the mutation rate parameter and scales the values added to the particular decision variables b and c (line 20). CR represents the crossover rate, used to combine weights between each parent and its child (line 21). $F \in [0, 1]$ and $CR \in [0,1]$ are determined by the user.

Therefore, the DE algorithm prevents the solution downside at a local extreme of the optimization function. In the DE algorithm the selection operator is efficient and fast, because it uses only two individuals (line 21) [5]. Application of DE for training of ANNs has been used in efficient manner with or without adaptive parameters (F and CR) [3], [9], [16], [17]. However, parameters tuning is a problem on ANN training, since F and CR can get fixed values or values from an adaptive function. User must decide over the manner of carrying the parameters F and CR. Besides defining parameters F and CR for mutation and crossover, DE also uses a tournament selection where the child vector competes against of its parent.

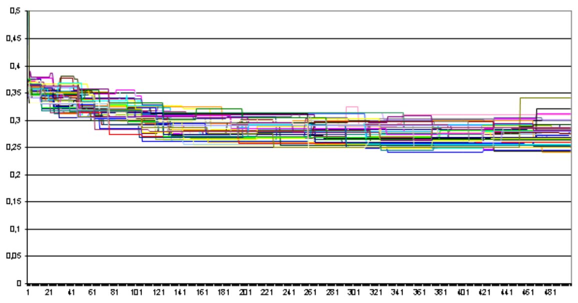
Figure 3 shows the typical behaviour of training evolution with global search methods (DE): (a) shows the improvement of both training and validation fitness in one of the selected best MLPs; (b) and (c) show how training and validation fitness evolutes in a 50-sized MLP population along 500 generations.



(a) MLP training (blue) and validation (red) fitness evolution.



(b) Training fitness evolution of MLP population.



(c) Validation fitness evolution of MLP population

Figure 3. Typical MLP training process with global search method

IV. BACKPROGAGATION (BP)

In supervised learning, training is performed by presenting a large set of examples, called the training set, to the network. Each example consists of a set of inputs presented to the input layer and the respective set of desired outputs presented to the output layer. Although training an ANN can be time-consuming, once this stage is successful completed, the input-output mapping is evaluated almost instantaneously. However, care must be taken to use an adequate training set, representative of the sampling space. In many cases this is not feasible, and the sampling space must be restricted to a specific sub-domain. This means that ANNs are best applied to specific well and defined problems [18].

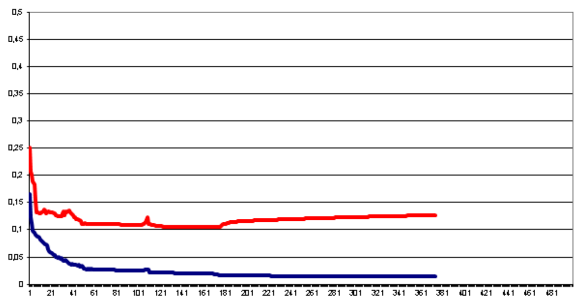
When using a MLP to solve a problem, the first activity is to train the MLP. Training depends on a strategic to choose initial weights and usually applies gradient learning algorithms to adapt weight values. Among these algorithms, error backpropagation (BP) method is one of the most used. In BP, the weight adjustment starts in the output nodes, where the measure of the error is available, and proceeds backpropagating this error through the previous layers.

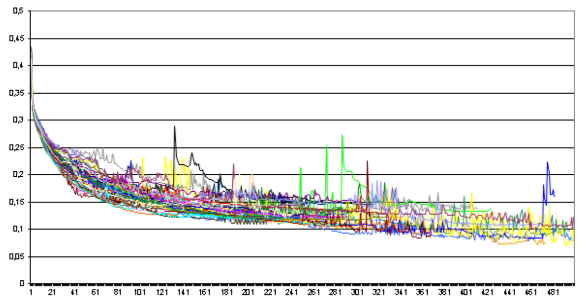
Proposed in 1986 by Rumelhart, Hinton and Willian, the error backpropagation method [10] is a learning procedure for multilayered feedforward neural networks. Basically the learning procedure is ruled in vectors, which are mapped as a set of inputs to a set of outputs. The mapping is specified by giving the desired activation state of output values for each presented state of input values.

Learning is carried out by iteratively adjusting the weights in the network so as to minimize the differences between the actual output state vector of the network and the target state vector. The network is initialized with random weights. During the learning process, an input vector is presented to the network and is propagated forward to determine the output signal. Next the output vector is compared with the target vector resulting in an error signal, which is backpropagated through the network in order to adjust weights. This process is repeated until the network responds for each input vector with an output vector that is sufficiently close to the desired one [13], [10]. In such process the activation of outputs is usually carried out by sigmoid function.

BP is a method based in gradient descent, that means BP does not assure to find a global minimum and can get stuck on local minima, where it will stay indefinitely. However BP is popular and widely used on ANN training.

Figure 4 shows the typical behaviour of training evolution with local search methods (BP): (a) shows Best the improvement of both training and validation fitness I one of the selected best MLPs – overfitting can be observed after cycle 180; (b) and (c) show how training and validation fitness evolutes in a 50-sized MLP population along 500 training cycles. Differences of local and global searches applied in MLP training can be clearly observed when comparing Figure 3 and Figure 4.


(a) MLP training (blue) and validation (red) fitness evolution


(b) Training fitness evolution of MLP population

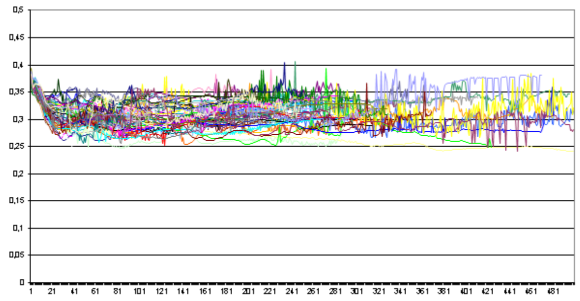

(c) Validation fitness evolution of MLP population

Figure 4. Typical MLP training process with local search method

V. Related Work

The use of evaluative algorithms has excelled to problem solving that requires space of global search optimization in several types problems. Theses algorithms have also been used to train ANNs, for example, DE algorithm. In 2003 Ihonen at al. [3] found in their experiments that the features of DE facilitate the training artificial neural networks, mainly because of the convergence of the solution is faster and more flexible maintenance of tuning parameters.

Slonik and Baiko [5] used the DE in training the FF neural network for the classification problem *parity p.* The results were compared with results of BP algorithm and with two more approach results: EA-NNT method [5] and LM Leven-Marquart algorithm [21]. However, in this comparison, DE proved a good alternative, primarily for data classification. In an attempt to increase the use of DE in ANNs FF training, Slonik [9] combined the selection of the adaptive parameters with the multiple vector technique and comparing the results obtained to the results of the three approaches:BP, EA-NTT and LM, the results were satisfactory from DE, but the memory consumption was very high in comparison to other approaches.

Yang et al. [26] have used DE and BP for the prediction of surface roughness in turning operations. The results obtained from the DE-based ANN model were compared with the BP-based ANN. The error percentage is very close in both cases, but the convergence speed using DE is higher than using BP.

Donate et al. [27] have evaluated different methods to evolve neural networks architectures in timeseries forecasting, comparing differential evolution (DE) and genetic algorithms. DE reached better after 200 generations due to the larger variation in population leads to more varied search over solution space.

Recently the research of Kawan and Mansor [23] applied the Evolutionary Algorithms: Cuckoo Search (CS) [24], PSO and Guaranteed Convergence PSO [25] (CGPSO is a variant of PSO) in an experiment for the problem of training the MLP with objective of minimizing the quadratic error. As in this paper, Kawan and Mansor also used the dataset of *UCI machine learning repository* in their experiments. The result showed that CS had the advantage upon PSO and GCPSO.

VI. DESCRIPTION OF EXPERIMENTS

In this experiment, five different databases were used for MLP training. With the objective of comparing results obtained from different MLP training approaches: backpropagation (BP), two variations of DE algorithm, and also a hybrid combination of such algorithms were used.

A. Databases and MLP Configurations

The databases used for MLP training in this experiment are as follow[1] [19]: Cancer (Breast Cancer Wisconsin Original Data Set), Diabetes (Pima Indians Diabetes Data Set), Glass (Glass Identification Data Set), Heart (Statlog Heart Data Set).

Each database was divided in 3 groups of instances, each one corresponding to training set, validation set, and testing set. These groups were set up with different sizes depending on the database as shown in Table I.

This in division was established a proportion of 50% to training dataset 25% to validation dataset and 25% to testing dataset, except to diabetes database in which the division was 33% to three instances.

TABLE I. TRAINING, VALIDATION AND TESTING SET INSTANCES

Database	Number of Instances		
	Training	*Validation*	*Testing*
Cancer	347	176	176
Diabetes	256	256	256
Glass	128	43	43
Heart	134	68	68
Iris	74	38	38

In the whole experiment some configurations were used by default. For all approaches, the MLPs were initialized with random weights between [-1, +1]. A hidden layer with different size was defined for each database. Cancer and heart databases used MLPs with 5 nodes in hidden layer. Diabetes, glass, and iris databases used MLPs with 10 nodes in hidden layer. Hidden and output nodes were connected to a bias with value 1. Connections between bias and all nodes were also weighted with random values between [-1, +1].

B. Training Approaches

Each database was used on the training process of a MLP population. We used 2 different population sizes (50 and 200), and for each population size, MLPs were trained with 4 different approaches, using 2 different algorithms:

- **BP:** backpropagation algorithm was used in each MLP and the best MLP element in the population was selected, considering the training cycle with best pair of training and validation fitness.
- **DE with fixed parameters**: as recommended in literature [22], the values of parameters F = 0.8 and CR = 0.5 were chosen.
- **DE with adaptive parameters**: F and CR were recalculated after each generation – 6 techniques were tested to select the method we consider more appropriate, and these techniques are described in session C.
- **Hybrid DE+BP:** DE with adaptive parameters was combined to BP, with the objective to join characteristics found in each one of these algorithms, i.e., the DE global search refined by the BP local search after each DE generation evolution. Fig. 3, shows part of DE algorithm (extract from Fig. 3) with a hybrid variation using backpropagation to local search over each global search improvement caught by DE. A BP call was added (lines 12a and 12b) every time a child Pi is generated and replaces its parent Qi in the population. Backpropagation runs up to maximum of 10 cycles and stops when Pi gets fitness improvement comparing to original Pi fitness generated by DE crossover

```
8:    …
9:    for each individual Pi ∈ P do
10:       AssessFitness(Pi)
11:       if Q ≠ {} and Fitness(Qi) <
                         Fitness(Pi) then
12:          Pi ← Qi
12a:      else
12b:         BackPragation(Pi);
13:       if Best={} or Fitness(Pi) <
                         Fitness(Best) then
14:          Best ← Pi
15:    Q ← P
16:    …
```

Figure 5. Partial hybrid DE algorithm with backpropagation

The evolution of each MLP element in the population is based only on the training fitness, but to select the best MLP to be used on the final testing phase, not only the training fitness was considered, but also the validation fitness. To be replaced for a new best MLP, the candidate MLP must have training and validation fitness both better than the corresponding ones in current best. Fig. 4, shows the MLP evolution of training fitness (blue) and validation fitness (red) along the maximum of 500 generations. Generation 187 was green lighted because it presented the best combination of training and validation fitness. Even better training fitness can be observed on next generations, there is no validation fitness improvement.

[1] http://archive.ics.uci.edu/ml/datasets.html

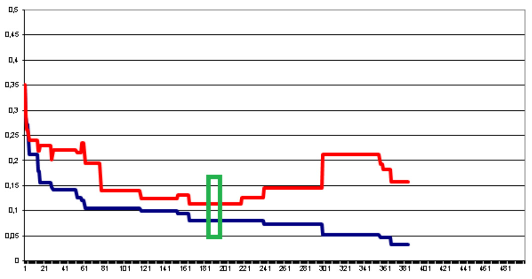


Figure 6. Best MLP Training Evolution

C. *Adaptive Parameters Techniques for DE*

As part of this experiment, we needed to select a method for dynamically changing F and CR parameters along the training process, We tested 6 different combinations of techniques, all with a population of 50 MLPs and using heart database. Partial results were compared and analyzed, and finally we chose the technique considered more appropriate to run as DE with adaptive parameters approach.

In a global adaptive adjustment of F and CR, these parameters are recalculated after each generation [9]. The A factor is used to identify the level of improvement each generation has in best partial fitness. [9] suggests A to be calculated as the ratio between the current generation (i) best fitness and the best fitness got in the previous generation (i-1), and then compute a new F and CR based on A and a random number chosen from 0 and 1. The new calculated values of F and CR are then applied for all MLP elements in population in the next generation.

$$A = \frac{BestFitness_i}{BestFitness_{i-1}} \quad (4)$$

$$F = 2 \times A \times random \quad (5)$$

$$CR = A \times random \quad (6)$$

Other techniques to adapt F and CR parameter were evaluated: a) adaptive element, where F and CR are computed for each element in the population; b) element adaptive with reduction factor, where a reduction factor of 0.5 is applied to A (A = A * 0.5) for each generation in that no improvement is reached on best fitness; c) global adaptive using best, where the best MLP in population is forced to be used in the linear combination; d) global adaptive using multiple vectors, where a vector of "n" candidates is randomly generated to choose one candidate to crossover (a vector of size 5 was used to test this approach) [9]; and e) element adaptive using best and reduction factor, that combines two of the techniques mentioned previously.

To compare the results of these 6 different techniques that calculate adaptive F and CR parameters, training and validation sets of heart database were used to measure not only the fitness, but also other parameters as: best training fitness, best validation fitness, generation which best fitness is found, scores of right answers using best MLP found with the validation set of instances, and cover rate considering a trust rate in the answers.

The approach "global adaptive with best and multiple vectors" had the best score in evaluation, but it shows an overload in CPU usage. Therefore, we decided to choose the second best approach, which is the one using only global adaptive parameters.

D. *Comparing BP and DE approaches*

BP, DE with fixed parameters, DE with adaptive parameters and a hybrid approach combining DE and BP were executed for each selected database, with populations of 50 and 200 individuals. The results were compared with different criteria as follow:

- **Best Fitness Generation**: tendency to reach the best fitness between the approaches BP, DE and BP+DE.
- **CPU Usage**: According to the training stop criterion previously defined, each population training execution could reach from 200 to 500 cycles or generations. CPU usage is a criterion that shows differences between how fast approaches can reach desired results the bests;
- **Best Fitness**: selection of best MLP considered both training and validation fitness as a criteria, as explained before (Fig. 6). Best fitness is the combination of training fitness and validation fitness obtained from the MLP chosen as best MLP during the training process;
- **Testing Best MLPs**: best MLPs were executed over testing group of instances for each database set. To demonstrate results obtained, some concepts used to calculate each MLP score: a) Trust Rate: responses obtained from MLP output layer nodes are given by float numbers between 0 and 1; desired answers are registered on databases as exactly 0 or 1 values for each instance class; de trust rate determines the degree of confidence that MLP output values should be considered as valid response to a positive classification; in the table below, trust rate used was 70%, which means only responses greater or equal to 0.7 were considered; b) Cover Rate: for some specific instances, MLP output values can be less than Trust Rate; in these cases, such instances are not covered by trust in their responses; Cover Rate means the number of instances which answers match Trust Rate over the total of instances considered in the MLP test; and c) Score: even when MLP outputs an answer that matches Trust Rate, it does not necessarily means the answer is right; MLP could classify an instance as if it belongs to Class1 with 99% of confidence when actually instance belongs to Class2; score measures the number of right answers given by the MLP comparing to desired answers registered on database, only based instances that match Rebiality Rate.

E. Results

According with Best Fitness Generation criteria, the Table II shows DE tends to take more time than BP to reach the best fitness (considering the generation number). When BP is combined with DE in the hybrid alternative, there is a tendency that the best fitness is reached before than in the solutions using only DE.

TABLE II. BEST FITNESS GENERATION

Databases	Best Fitness Generation		
	BP	***DE***	***DE+BP***
Cancer	63	307	92
Diabetes	31	113	103
Glass	59	49	155
Heart	288	95	225
Iris	152	184	101

(a) Population size = 50

Databases	Best Fitness Generation		
	BP	***DE***	***DE+BP***
Cancer	276	134	26
Diabetes	3	453	320
Glass	7	282	130
Heart	231	184	53
Iris	53	480	122

(b) Population size = 200

Because of backpropagation is not exactly an evolution algorithm, we used a group of independent MLPs and selected the best MLP from this group after the training processes finishes. The number of MLPs being trained with BP was the same of population size used with DE. The maximum of 500 cycles for BP and 500 generations for DE were defined as timeout in the training process. However, when 200 cycles or generations are reached with no improvement on best, the training process also finishes.

According to the CPU usage criteria, Table III shows that BP is faster than DE approaches for all selected databases.

TABLE III. CPU USAGE

Approach	CPU Usage (minutes)				
	Cancer	***Diabetes***	***Glass***	***Heart***	***Iris***
BP	33	41	36	34	25
DE fixed	56	216	131	97	40
DE adaptive	81	174	168	78	151
DE + BP	58	47	38	64	35

(a) Population size = 50

Approach	CPU Usage (minutes)				
	Cancer	***Diabetes***	***Glass***	***Heart***	***Iris***
BP	53	76	65	59	26
DE fixed	184	343	196	185	82
DE adaptive	236	365	139	293	36
DE + BP	174	165	116	187	37

(b) Population size = 200

Data presented in Table III were obtained after one execution of each algorithm and variation. Next, we chose 3 databases (glass, heart and iris) to be executed 10 times with BP and DE+BP approaches and with a population size of 50. Table IV shows the average and standard deviation of CPU usage. Values are similar to the ones obtained previously with only one execution, and confirm that BP is relative faster than approaches using DE.

TABLE IV. AVERAGE AND STANDARD DEVIATION CPU USAGE

Approach	CPU Usage (minutes)			Standard Deviation		
	Glass	***Heart***	***Iris***	***Glass***	***Heart***	***Iris***
BP	36	35	13	3,52	2,59	1,14
DE + BP	55	76	17	3,35	2,09	1,14

According to the best fitness criteria obtained from the best MLP chosen along the population training process for each database, each population size and each algorithm approach, results can be compared in Table V. Best training and validation fitness for each database was bold highlighted.

TABLE V. BEST TRAINING VALIDATION FITNESS (FROM BEST MLP)

Algorithm	Training Fitness				
	Cancer	***Diabetes***	***Glass***	***Heart***	***Iris***
BP	**0.003**	**0.110**	**0.057**	**0.018**	0.015
DE fixed	0.047	0.169	0.370	0.091	0.140
DE adaptive	0.022	0.166	0.372	0.097	0.100
DE + BP	0.021	0.141	0.176	0.032	**0.013**

(a) Training Fitness for Population size = 50

Algorithm	Training Fitness				
	Cancer	***Diabetes***	***Glass***	***Heart***	***Iris***
BP	**0.004**	**0.110**	**0.071**	**0.018**	0.015
DE fixed	0.028	0.195	0.367	0.053	0.159
DE adaptive	0.026	0.146	0.297	0.081	0.033
DE + BP	0.020	0.125	0.166	0.049	**0.014**

(b) Training Fitness for Population size = 200

Algorithm	Validation Fitness				
	Cancer	*Diabetes*	*Glass*	*Heart*	*Iris*
BP	0.024	0.194	**0.246**	**0.094**	**0.018**
DE fixed	0.024	0.204	0.403	0.116	0.117
DE adaptive	**0.022**	0.177	0.319	0.114	0.034
DE + BP	0.025	**0.174**	0.262	0.106	0.021

(c) Validation Fitness for Population size = 50

Algorithm	Validation Fitness				
	Cancer	*Diabetes*	*Glass*	*Heart*	*Iris*
BP	0.021	0.194	**0.213**	**0.094**	0.018
DE fixed	0.052	0.195	0.366	0.116	0.126
DE adaptive	**0.019**	0.181	0.371	0.125	0.093
DE + BP	0.023	**0.176**	0.253	0.095	**0.017**

(d) Validation Fitness for Population size = 200

BP algorithm reached the best values for training fitness except for iris database, even considering two different population sizes, and BP also reached most of best validation fitness.

Considering test phase of best MLPs, Table VI shows Scores and Cover Rates obtained from a Trust Rate of 70%. These measures were calculated after one execution training for each database, each population size of 50 and 200, and each algorithm used.

TABLE VI. COMPARATIVE OF SCORES (TRUST RATE = 70%)

Algorithm	Cancer		Diabetes		Glass		Heart		Iris	
	Score	*Cover*	*Score*	*Cover*	*Score*	*Cover*	*Score*	*Cover*	*Score*	*Cover*
BP	95.9	99.4	70.2	86.0	81.3	81.6	80.3	97.0	94.7	100
DE fixed	93.1	100	50.0	4.6	84.6	43.3	67.1	100	96.5	76.3
DE adaptive	95.2	97.1	100	16.2	89.8	57.8	83.3	97.0	94.5	97.3
DE + BP	97.6	98.2	77.2	51.1	86.1	62.1	84.1	92.6	94.7	100

(a) Population size = 50

Algorithm	Cancer		Diabetes		Glass		Heart		Iris	
	Score	*Cover*	*Score*	*Cover*	*Score*	*Cover*	*Score*	*Cover*	*Score*	*Cover*
BP	96.5	98.8	81.3	81.6	81.2	74.4	80.3	97.0	94.7	100
DE fixed	94.1	97.7	83.2	53.5	42.8	16.2	79.7	100	100	50.0
DE adaptive	94.7	97.7	88.8	52.7	75.0	18.6	82.3	100	100	81.5
DE + BP	98.2	95.4	85.8	69.1	62.8	81.4	83.0	95.5	94.7	100

(b) Population size = 200

On the results shown in Table VI, we can observe that DE+BP seems to be a little more efficient than using only BP, but in the statistical results presented in Table VII both algorithms are equivalent considering scores and BP has a small advantage compare to DE+BP considering the cover rate.

Data presented in Table VI were calculated based in only one execution of each algorithm for each database. To confirm data obtained we chose 3 databases (glass, heart and iris) to be executed 10 times with BP and DE+BP approaches and with a population size of 50. Table VII shows the average of scores and cover rated calculated from 10 executions.

TABLE VII. AVERAGE SCORES AND COVER RATES

Algorithm	BP		DE+BP	
	Score	*Cover*	*Score*	*Cover*
Glass	71.4	72.0	71.3	65.5
Heart	82.5	92.6	82.2	83.6
Iris	94.74	100	94.75	99.2

Identify the statistic significance between the values that compose the median *score* and *cover* obtained from the Glass, Heart and Iris bases (Table VII), the statistical method Kruskal-Wallis was used by way of the R-Commander tool [20]. Only the values of the average *score* for the Heart base presented statistic significance of 0.01341, in other words, *p-value*=0.01341.

VII. CONCLUSION

Based on results contained in the Tables II-VII, we can observe that DE tends to take more time than BP to reach the best fitness. DE improves the time spent to reach the best fitness when combined with BP. We can also observed that the training execution with the BP algorithm takes less time consuming than DE algorithm. This makes sense since the DE algorithm needs to create more MLPs instances to perform a linear combination. Considering training and validation the *fitness,* despite the apparent advantage presented by BP algorithm relative to DE algorithm, statistical test showed that both algorithms are equivalent to measured *scores,* although the BP is better to cover desired confidence tests. The DE algorithm provides the advantage of global research and may also be combined with others algorithms in hybrid solutions for best results. Nevertheless, this BP algorithm experiment was more efficient to cover the cases used in test based on the confidence parameter used. Future work is suggested for testing other hybrid variations to combine global and local search algorithms and adjust the parameters used in attempting to find more efficient solutions for the classification of this data and others.

REFERENCES

[1] Y. Shang and W. W. Benjamin, "Global Optimization for Neural Network Training", IEEE Computer, Vol. 29, no.3, pp. 45-54, March 1996.

[2] S. Luke, "Essentials of Metaheuristics", Department of Computer Science, George Mason University, Online Version 1.1, January, 2011.

[3] K. Ilonen, Joni-Krisitan and J. Lampinen, “Differential Evolution Training Algorithm for Feed-Forwar Neural Networks”, Neural Processing Letters, Kluwer Academic Publishers, Vol.17, No.1 pp 93-105, March 2003

[4] S. S. Haykin, “Redes neurais: principios e prática”, 2ª.ed. Porto Alegre, RS Bookman, 2001.

[5] A. Slowik and M. Bialko, “Training of Artificial Neural Networks Using Differential Evolution Algorithm”, *IEEE, HIS*, pp. 60-65, Krakow, Poland, May, 2008.

[6] J. Kennedy and R. Eberhart, “Particle Swarm Optimization, Neural Networks”, IEEE International Conference, Vol. 4, pp. 1942-1948, 1995.

[7] R. Storn and K. Price “Differential Evolution - A simple and Efficient Heuristic for Global Optimization over Continuous Spaces”, Journal of Global Optimization, Vol. 11, pp 341-359, Netherlands, 1997.

[8] D. Karaboga and S. Ökdem “A simple and global optimization algorithm for engineering problems: Diferential Evolution Algorithm”, Turk J Elec Engin, Vol.12 No.1. 2004.

[9] A. Slowik, “Applications of an Adaptive Differential Evolution Algorithm With Multiple Trial Vectors to Artificial Neural Network Training”, IEEE Transactions on Industrial Electronics, Vol. 58, no.8, pp. 3160-3167 August, 2011.

[10] D. E. Rumelhart, G. E. Hinton and R. J. Williams, “Learning representations by back-propagating errors”, Nature, Vol. 323 pp 533- 536, October 1986.

[11] C. Blum and K. Socha, “Training feed-forward neural networks with ant colony optimization: An application to pattern classification”, HIS`05, Fifth Intern. Confer. on Hybrid Intelligent Systems, IEEE Computer Society Washington, USA, pp., 233-238, 2005.

[12] D. Karaboga, B. Akay and C. Ozturk, “Artificial Bee Colony (ABC) Optmization Algorithm for Training Feed-Forward Neural Networks”, Modeling Decision for Artificial Intelligence, Springer Berlin/Heidelberg, Vol. 4617, pp. 318-329, 2007.

[13] A. V. Ooyen and B. Nienhuis, Improving the Convergence of the Backpropagation Algorithm, Neural Networks, vol.5, pp. 465-471, Pergamon Press, 1992.

[14] D. C. Plaut, S. J. Nowlan and G. E. Hinton, “Experiments on Learning by Back Propagation”, Carnegie-Mellon University, Pittsburgh, PA 15213, Technical Report CMU-CS86,126, June 1986.

[15] I. Yilmaz and O. Kaynar, “Multipe regression, ANN (RBF, MLP) and ANFIS models for prediction of swell potential of clayey soils”, Expert Systems with Applications, Elsiver Vol.38, pp 5958-5966, 2011.

[16] N. G. Pavlidis, D. K. Tasoulis, V. P. Plagianakos and M. N.Vrahatis , “Spiking neural network training using evolutionary algorithms”, IEEE, Neural Networks, Vol. 4 pp. 2190-2194 Aug., 2005.

[17] H. M. Abdul-Kaner, “Neural Networks Training Basead on Differentisl Evolution Algorithm compared Other Architectures for Weather”, I JCSNS International Journal of Computer Science and Network Security, Vol.9, No.3, pp 92-99, Mar. 2009.

[18] A. Gaspar-Cunha and A. Vieira, “A Multi-Objective Evolutionary Algorithm Using Neural Networks to Approximate Fitness Evaluations”, International Journal of Computers, Systems and Signals, Vol.6, No. 1, pp 18-36, 2005.

[19] A. Frankand and A. Assuncion, “UCI Machine Learning Repository” http://archive.ics.uci.edu/ml, Irvine, CA: University of California, School of Information and Computer Science, 2010.

[20] R-Project, “The R Project for Statistical Computing”, www.r-project.org, June 2012.

[21] D.W Marquardt, “An algorithm for least-squares estimation of nonlinear parameters”, Journal of the Society for Industrial and Applied Mathematics, Philadelphia, v. 11, n. 2, p. 431–441, 1963.

[22] J. Tvrd´ık, “Adaptive Differential Evolution: Application to Nonlinear Regression”. Proceedings of the International Multiconference on Computer Science and Information Technology pp. 193–202, 2007.

[23] A. Al. Kawam and N. Mansour, “Metaheuristic Optimization Algorithms for Traininh Artificial Neural Networks” International Journal od Computer and Information Tecnology – ISSN: 2279-0764 vol.1, pp. 156-161, November 2012,

[24] XS Yang, S. Deb. “Cuckoo search via Lévy flights”. Proc. of World Congress on Nature & Biologically Inspired Computing. 2009. pp 210-214.

[25] F. Van den Bergh and A. P. Engelbrecht, “A new locally convergent particle swarm optimizer”, *Proc. IEEE Conference on Systems, Man and Cybernetics* (Hammamet Tunisia), Vol. 3, 6-9 Oct, 2002.

[26] S. H. Yang, U. Natarajan, M. Sekar and S. Palani, “Prediction of surface roughness in turning operations by computer vision using neural network trained by differential evolution algorithm” Spring-Verlag, London, pp.965-971 Apr, 2010.

[27] J. P. Donate, X li, G. G. Sanchez and A. S. de Miguek “Time series forecasting by evolving artificial neural networks with genetic algorithms, differential evolution and estimation of distribution algorithm”, Springer Verlag, London, Neural Comput & Applic, Sep 2011.