Evolving Neural Networks: A Comparison between Differential Evolution and Particle Swarm Optimization

Beatriz A. Garro[1], Humberto Sossa[1], and Roberto A. Vázquez[2]

[1] Centro de Investigación en Computación – IPN
Av. Juan de Dios Bátiz, esquina con Miguel de Othón de Mendizábal
Ciudad de México, 07738, México
[2] Intelligent Systems Group, Facultad de Ingeniería – Universidad La Salle
Benjamín Franklin 47 Col. Condesa CP 06140 México, D.F
bgarrol@ipn.mx, hsossa@cic.ipn.mx,
ravem@lasallistas.org.mx

Abstract. Due to their efficiency and adaptability, bio-inspired algorithms have shown their usefulness in a wide range of different non-linear optimization problems. In this paper, we compare two ways of training an artificial neural network (ANN): Particle Swarm Optimization (PSO) and Differential Evolution (DE) algorithms. The main contribution of this paper is to show which of these two algorithms provides the best accuracy during the learning phase of an ANN. First of all, we explain how the ANN training phase could be seen as an optimization problem. Then, we explain how PSO and DE could be applied to find the best synaptic weights of the ANN. Finally, we perform a comparison between PSO and DE approaches when used to train an ANN applied to different non-linear problems.

1 Introduction

A feed-forward artificial neural network (ANN) is a powerful tool widely used in the field of pattern recognition and time series analysis. However, despite their power in some practical problems, ANNs cannot reach an optimum performance in several non-linear problems. This fact is caused because the parameters, used during learning phase such as learning rate, momentum, among others, do not allow computing the best set of synaptic weights. In the field of the evolutionary computation, there are many algorithms that allow obtaining an optimum (or more) of a specific problem. One of these techniques is the Differential Evolution (DE) algorithm, based on the classical steps of the Evolutionary Computation. DE performs mutation based on the distribution of the solutions in the current population. In this way, search directions and possible step-sizes depend on the location of the individuals selected to calculate the mutation values.

On the other hand, the Particle Swarm Optimization (PSO) algorithm is inspired by observations of social interaction. PSO operates on a population of *particles* (that are the individuals), evolving them over a number of iterations with the goal of finding a solution to an optimization function. This metaphor searches an optimum solution based on the self and social experience of the best particle of the population.

Y. Tan et al. (Eds.): ICSI 2011, Part I, LNCS 6728, pp. 447–454, 2011.

Several works that use evolutionary strategies for training ANNs have been reported in the literature. For example, in [1], the authors combine PSO and ANNs for function approximation. Another application presented in [2] is a PSO-based neural network in the analysis of outcomes of construction claims in Hong Kong. Other examples can be found in [3], [4], [5], [6], [7] and [8].

DE algorithm has been less used in this kind of work. For example, Ilonen et al [9], propose the learning of an ANN by finding the synaptic weighs through the classical DE. Other examples can be found in [10] and [11]. In [12], it shows a large literature review where the evolutionary algorithms are used to evolve the synaptic weights.

In this paper, we compare the accuracy of PSO and DE algorithms during training the synaptic weights of ANN when applied to solve different classification problems. All of this, in order to determine which algorithm is better in the task of adjusting the synaptic weights of an ANN. This comparative is important because these two algorithms have got a different scheme each one. The way to manipulate the data for doing a new population is the key to find a good solution. The task of training a set of synaptic weights for an ANN is an essential aim and using these two algorithms when observing the ANN behavior. The aim of this paper is to explain how the neural network training phase could be seen as an optimization problem and how PSO and DE could be applied to find the best synaptic weights of the ANN. In the next section, we present a basic definition of an ANN. In section 3 basic PSO algorithm functioning is explained, while section 4 is devoted for DE algorithm. In section 5, we explain how PSO and DE could be applied to find the best synaptic weights of the ANN. In section 6 we perform a comparison between several ANNs trained by means of PSO and DE algorithms, when applied to solve different non-linear problems. In section 7, we finally give the conclusions of those experimental results.

2 Basics on Feed-Forward Neural Networks

A basic description of an ANN can be: a massively parallel-distributed processor made up from simple processing units. This type of processing unit performs in two stages: weighted summation and some type of nonlinear function. Each value of an input pattern $\mathbf{A} \in \mathbb{R}^N$ is associated with its weight value $\mathbf{W} \in \mathbb{R}^N$, which is normally between 0 and 1. Furthermore, the summation function often takes an extra input value θ with weight value of 1 to represent threshold or *bias* of a neuron. The summation function will be then performed as,

$$y = f\left(\sum_{i=1}^{N} a_i w_i + \theta\right) \tag{1}$$

The sum-of-product value is then passed into the second stage to perform the activation function $f(x)$ which generates the output from the neuron and determines the behavior of the neural model. In a multilayer structure the input nodes, which received the pattern $\mathbf{x} \in \mathbb{R}^N$, the units in the first hidden layer, then the outputs from the first hidden layer are passed to the next layer, and so on until they reach the output layer and produce an approximation of the desired output $\mathbf{y} \in \mathbb{R}^M$.

Basically, learning is a process by which the free parameters (i.e., synaptic weights $\mathbf{W}$ and bias levels θ) of an ANN are adapted through a continuous process using a labeled set of training data made up of p input-output samples:

$$\mathbf{T}^{\xi} = \left\{ \left(\mathbf{x}^{\xi} \in \mathbb{R}^{N}, \mathbf{d}^{\xi} \in \mathbb{R}^{M} \right) \right\} \forall \xi = 1, \ldots, p \tag{2}$$

where $\mathbf{x}$ is the input pattern and $\mathbf{d}$ the desired response.

Given the training sample $\mathbf{T}^{\xi}$, compute the free parameters of the neural network so that the actual output $\mathbf{y}^{\xi}$ of the neural network due to $\mathbf{x}^{\xi}$ is close enough to $\mathbf{d}^{\xi}$ for all ξ in a statistical sense. In this sense, we might use the mean-square error given in eq. 3 as the objective function to be minimized:

$$e = \frac{1}{p \cdot M} \sum_{\xi=1}^{p} \sum_{i=1}^{M} \left(d_i^{\xi} - y_i^{\xi} \right)^2 \tag{3}$$

3 Basics on Particle Swarm Optimization

PSO algorithm is a method for the optimization of continuous non-linear functions proposed by James Kennedy and Russell C. Eberhart. This algorithm is inspired on observations of social and collective behavior as well as fish schooling or bird flocking [13]. For example, a population or a flock could be considered as a cumulus of particles i where each particle represents the position $\mathbf{x}_i$ of a particle in a multidimensional space. These particles (individuals) also represent a possible solution of a specific function optimization. According to a velocity function $\mathbf{v}_i$ which takes into account the best position of a particle in a population $\mathbf{p}_g$ (i.e., social component) as well as the own best position of the particle $\mathbf{p}_i$ (i.e., cognitive component) the particles will move each iteration to a different position until they reach an optimum position. At each time step t, the velocity of a particle i is updated using

$$\mathbf{v}_i(t+1) = \omega \mathbf{v}_i(t) + c_1 r_1 \left(\mathbf{p}_i(t) - \mathbf{x}_i(t) \right) + c_2 r_2 \left(\mathbf{p}_g(t) - \mathbf{x}_i(t) \right) \tag{4}$$

where ω is the inertia weight and typically setup to vary linearly from 1 to near 0 during the course of an iteration run; c_1 and c_2 are acceleration coefficients; $r_1 \sim U(0,1)$ and $r_2 \sim U(0,1)$ are uniformly distributed random numbers in the range $(0,1)$. The velocity $\mathbf{v}_i$ is limited to the range $[v_{\min}, v_{\max}]$. Updating the velocity in this way enables the particle i to search around its individual best position $\mathbf{p}_i$, and the global best position $\mathbf{p}_g$. Based on the updated velocities, the new position of the particle i is computed using

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1). \tag{5}$$

Finally, this optimum position will be the solution which maximizes or minimizes an objective function.

The basic PSO algorithm for the real version could be performed as follows:

Given a population of $\mathbf{x}_i \in \mathbb{R}^D, i = 1,\ldots,M$ individuals

1) Initialize the population at random
2) Until a stop criterion is reached:
   a) For each individual $\mathbf{x}_i$ evaluate their fitness.
   b) For each individual i, update its best position $\mathbf{p}_i$.
   c) From all individual i, update the best individual $\mathbf{p}_g$.
   d) For each individual i, compute the velocity update equation $\mathbf{v}_i(t+1)$ and then compute the current position $\mathbf{x}_i(t+1)$.

4 Basics on Differential Evolution

In 1995 an adaptive and efficient scheme emerged: Differential Evolution algorithm, proposed by Kenneth Price and Rainer Storn, useful for Global Optimization over continuous spaces [14]. With this precedent, it was opened a new optimization technique in Evolutionary Computation. Due to its exploration capacity over a search space of a given problem, the DE algorithm avoids staying in a local optimum. It has few parameters and it converges to the optimum faster than others evolutionary techniques (the solution's representation is given by vectors of real numbers). All these characteristics convert DE into an excellent optimization algorithm of a complex, non-differential and non-continuous problems.

The pseudo code of "DE/rand/1/bin" is shown in the next algorithm adapted from [15].

1. Randomly select two vectors from the current generation.
2. Use these two vectors to compute a difference vector.
3. Multiply the difference vector by weighting factor F.
4. Form the new trial vector by adding the weighted difference vector to a third vector randomly selected from the current population.

5 Evolving the Synaptic Weights of an ANN Using PSO and DE

In this section, we describe how given a set of patterns $\mathbf{T}$, the synaptic weights of an ANN can be automatically adjusted by means of a basic PSO and DE. The architecture of the ANN has to be previously defined because we only evolve the synaptic weights and the architecture must be static. The synapses weights of the ANN are codified based on a vector that represents a graph $\mathbf{x}$. The vector is a solution composed with the set of synaptic weights $w_{i,j}^k$ between neuron i and neuron j that belongs to the layer k, until a maximum number of neurons MNN. This vector represents an individual (particle or solution) of the population that will be evolved by PSO. The individual (see Fig. 1) has a set of values that change with respect to time and each

individual is evaluated in the fitness function in order to calculate the minimum square error generated by the ANN which maximizes the performance. The individual whose weights provoke the minimum value of the MSE will be the best solution for the trained ANN. This individual's representation is used by the two algorithms: PSO and DE.

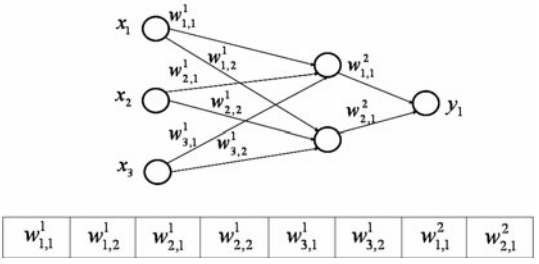

Fig. 1. Representation of an individual composed of a set of synaptic weights

Due to we are working with predefined feed-forward architectures, the fitness function which measures the performance of an individual is given by eq. 3 where the output y_i of the ANN is computed by means of equation 1.

6 Experimental Results

In order to evaluate the accuracy of the PSO and DE algorithms, several experiments were performed. Three well-known data-sets were taken from the UCI machine learning benchmark repository [16] to test the algorithms: iris plant, wine and breast cancer datasets. Also a real object recognition problem composed of 100 images of five different objects was used.

All datasets were randomly partitioned into two sets: 50% for the training data and 50% for the testing data. The input features of all data set were rescaled in a range between $[0,1]$ and the outputs were encoded by a winner-take-all representation. There are two conditions for the algorithm to stop: when it achieves the total number of generations, and when the algorithm achieves a $MSE = 1\times10^{-10}$. This last condition can be replaced using a validation set.

Before starting with the experiments, we defined the parameters of the three algorithms. For the case of the basic PSO algorithm, the population size M was set to 50, the number of generations was set to 4000, the initial position of the particles was in the range $[-40,40]$, velocity range $[-2,2]$, $\omega = 0.729$, $c_1 = c_2 = 2$. For the case of the basic DE algorithm, the population size $NP = 50$, number of generations was set to 4000, the population was initialized in the range $[-40,40]$, $CR = 0.9$, and $F = rand[0.3,0.7]$.

These parameters were set to the same value for all the experiments according to the literature (ω, c_1, c_2, F and CR). The others were defined by means of our criteria

and considering the limits of the numeric representations in the computer. The velocity range is different from 0 and not so big, in order to obtain possible directions without big displacements that can affect particle's movement. Before starting to evolve the synaptic weights, it was necessary to determine the architecture of the ANN. In this case, we decided to use an ANN with three layers: the input, the output and one hidden layer compose of five neurons.

Ten experiments for each dataset were performed to measure the accuracy of the proposed method. Figs. 2 and 3 show the percentage of recognition achieved with the ANN applied to solve the four problems by means of PSO and DE. The two algorithms found the best sets of synaptic weights that minimized the MSE of the ANN. After that, the classification error (CER) was calculated.

Fig. 2 shows the accuracy of the ANN trained with the PSO algorithm. As it can be observed from this figure, the best recognition percentages were achieved for the Iris plant and for the breast cancer data set, see Fig. 2 (a) and (c) respectively. For the case of the object recognition problem using PSO, the percentage of recognition was not as good as desired (Fig. 2 (d)). Furthermore, for the case of the wine data set, the percentage of recognition achieved by the ANN was the worst.

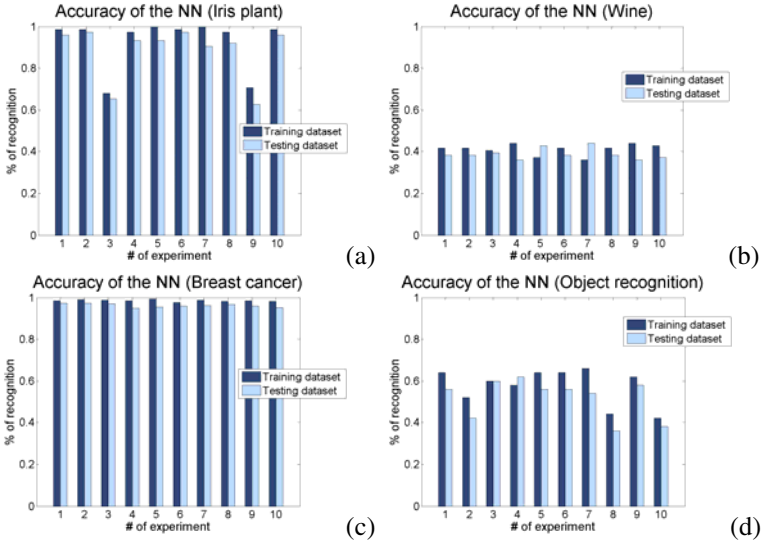

Fig. 2. Accuracy of the ANN using PSO algorithm. (a)Iris plant data set. (b)Wine data set. (c)Breast cancer data set. (d)Real object recognition data set.

Fig. 3 shows the accuracy of the ANN trained with the DE algorithm. Using this algorithm we can appreciate that the percentage of recognition for all the data sets was much better than the results obtained by PSO algorithm. Besides this, for the real object recognition problem, see Fig. 3(d), DE algorithm could obtain a percentage of recognition over the 90% in comparison with other results (Fig.3 (a), (b) and (c)). In addition to this, from Table 1 the reader can appreciate the average classification error (CER) for all experimental results just as the standard deviation for training phase and

testing phase. The results show that the best technique for evolving the synaptic weights of an ANN was the DE algorithm. This could be due to PSO algorithm has more parameters to tune than DE algorithm.

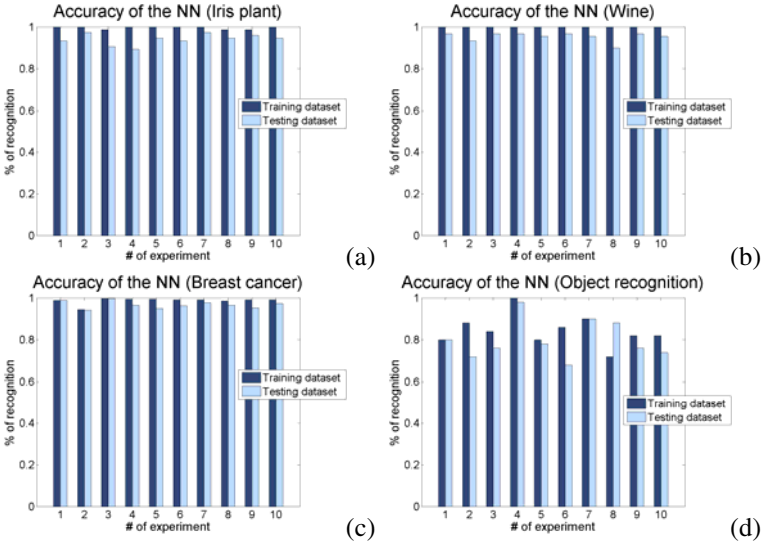

Fig. 3. Accuracy of the ANN using DE algorithm. (a)Iris plant data set. (b)Wine data set. (c)Breast cancer data set. (d)Real object recognition data set.

Table 1. Average and standard deviation applying the CER

Data base	PSO algorithm		DE algorithm	
	Tr. Er.	Te. Er.	Tr. Er.	Te. Er.
Iris plant	0.072 ± 0.124	0.116 ± 0.130	0.004 ± 0.006	0.058± 0.026
Wine	0.589 ± 0.026	0.612 ± 0.026	0 ± 0	0.047 ± 0.021
Breast Cancer	0.014 ± 0.026	0.037 ± 0.008	0.013 ± 0.015	0.032 ± 0.017
Object Recognition	0.424 ± 0.086	0.482 ± 0.094	0.156 ± 0.074	0.2 ± 0.092

Tr. Er = Training Error, Te. Er. = Testing Error.

7 Conclusions

In this paper, we compared two powerful bio-inspired algorithms in order to determine which one is more suitable to train an ANN: PSO and DE. This is very important because the training of an ANN is one of the keys issues to obtain a good generalization, and it is necessary to know the behavior of the evolutionary algorithms in the basic design of an ANN.

We explained in detail how the ANN's training phase could be seen as an optimization problem. Then, it was explained how PSO and DE could be applied to find the optimal synaptic weights of the neural network. Finally, we performed a comparison between a neural network trained with the PSO and DE algorithms, when applied to solve different non-linear pattern classification problems.

Through several experiments, we observed that DE algorithm was better in searching the best set of synaptic weights. This does not mean that PSO is not useful in this task, but it requires to determine the best set of parameters that improve its performance. For future woks, we need to test the experimentation applying PSO algorithm with neighborhoods due to this improvement avoid to be trapped in a local minimums.

Acknowledgements. H. Sossa thanks SIP-IPN, COTEPABE and DAAD-PROALMEX for economical support. Authors also thank the European Union, the European Commission and CONACYT for the economical support. This paper has been prepared by economical support of the European Commission under grant FONCICYT 93829. The content of this paper is an exclusive responsibility of the CIC-IPN and it cannot be considered that it reflects the position of the European Union.

References

[1] Tejen, S., Jyunwei, J., Chengchih, H.: A Hybrid Artificial Neural Networks and Particle Swarm Optimization for Function Approximation. International Journal of Innovative Computing, Information and Control ICIC 4, 2363–2374 (2008)
[2] Chau, K.W.: Application of a PSO-based neural network in analysis of outcomes of construction claims. Automation in Construction 16, 642–646 (2007)
[3] Chatterjee, A., et al.: A Particle Swarm Optimized Fuzzy-Neural Network for Voice Controlled Robot Systems. IEEE Trans. on Ind. Elec. 52, 1478–1489 (2005)
[4] Wang, Z., et al.: Particle Swarm Optimization and Neural Network Application for QSAR. In: Proceedings 18th Parallel and Distributed Processing Symposium (2004)
[5] Zhao, L., Yang, Y.: PSO-Based Single Multiplicative Neuron Model for Time Series Prediction. Expert Systems with Applications 36, 2805–2812 (2009)
[6] Da, Y., Ge, X.R.: An improved PSO–based ANN with simulated annealing technique. Neurocomput. Lett. 63, 527–533 (2005)
[7] Yu, J., et al.: An Improved Particle Swarm Optimization for Evolving Feedforward Artificial Neural Networks. Neural Processing Letters 26, 217–231 (2007)
[8] Mohaghegi, S., et al.: A comparison of PSO and backpropagation for training RBF neural networks for identification of a power system with STATCOM. In: Proceedings 2005 IEEE Swarm Intelligence Symposium on SIS 2005, pp. 381–384 (2005)
[9] Ilonen, J., Kamarainen, J.-K., Lampinen, J.: Differential Evolution Training Algorithm for Feed-Forward Neural Networks. Neural Processing Letters 17(1), 93–105 (2003)
[10] Castillo, P.A., Carpio, J., Merelo, J.J., Prieto, A., Rivas, V.: Evolving multilayer perceptrons. Neural Process. Lett. 12, 115–127 (2000)
[11] Rivero, D., Rabuñal, J.R., Dorado, J., Pazos, A.: Automatic design of aNNs by means of GP for data mining tasks: Iris flower classification problem. In: Beliczynski, B., Dzielinski, A., Iwanowski, M., Ribeiro, B. (eds.) ICANNGA 2007. LNCS, vol. 4431, pp. 276–285. Springer, Heidelberg (2007)
[12] Yao, X.: Evolving Artificial Neural Networks. Proc. of IEEE 87(9), 1423–1446 (1999)
[13] Kennedy, J., Eberhart, R.C., Shi, Y.: Swarm Intelligence. Morgan Kaufmann Publishers, San Francisco (2001)
[14] Storn, R., Price, K.: Differential Evolution - a Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces, TR-95-012, ICSI (1995)
[15] Mezura-Montes, E., Velazquez-Reyes, J., Coello Coello, C.A.: A Comparative Study of Differential Evolution Variants for Global Optimization. In: GECCO (2006)
[16] Murphy, P.M., Aha, D.W.: UCI repository of machine learning databases. Dept. Inf. Comput. Sci., Univ. California, Irvine, CA (1994)