Design of Artificial Neural Networks using a Modified Particle Swarm Optimization Algorithm

Beatriz A. Garro, Humberto Sossa and Roberto A. Vazquez

Abstract— **In the last years, bio-inspired algorithms have shown their power in different non-linear optimization problems. Due to the efficiency and adaptability of bio-inspired algorithms, in this paper we explore a new way to design an artificial neural network (ANN). For this task, a modified PSO algorithm was used. We do not only study the problem of finding the optimal synaptic weights of an ANN but also its topology and transfer functions. In other words, given a set of inputs and desired patterns, with the proposal we are able to find the best topology, the number of neurons, the transfer function for each neuron, as well as the synaptic weights. This allows to designing an ANN to be used to solve a given problem. The proposal is tested using several non-linear problems.**

I. Introduction

NEURAL networks are powerful tools widely used in the field of pattern recognition, computer vision and time series analysis. Perhaps, among the most popular neural network models we could mention the feed-forward neural network trained with the back-propagation algorithm. However, in despite of their powerful in some practical problems, neural networks can not reach an optimum performance in non-linear problems. This fact is caused because the parameters used during learning phase such as learning rate, momentums, among others, do not allow compute the optimums synaptic weights.

On the other hand, a very new interesting field were bio-inspired algorithms can be applied are in optimization problems. These algorithms are based in natural behaviors. Among these algorithms we could mention the Particle Swarm Optimization (PSO) algorithm. The particle swarm optimization algorithm is inspired by observations of social interaction. A PSO operates on a population of *particles*, evolving them over a number of iterations with the goal of finding a solution to an optimization function. This metaphor allows to search an optimum solution based on the own experience and a social experience of the best particle of the population.

Several investigations to the one reported in this paper can be found in the literature. Evolutionary algorithms (EAs) are non-gradient approaches, and are very promising approaches for training ANNs. In [23], Xin Yao shows a considerable large literature review where EAs are used to evolve ANNs; refer for example to [15], [16] and [22]. However, few works in the evolution of ANNs using swarm optimization have been done. For example, in [1], the authors combine PSO and ANNs for the function approximation problem. Other application presented in [2] is a PSO-based neural network in the analysis of outcomes of construction claims in Hong Kong. In [3], the authors use PSO in fuzzy-neural networks for voice-controlled robot systems. In [4], a PSO technique is used to select the most important characteristics from a set of patterns; it then uses a back propagation neural network for the construction of QSAR models. In general, the authors present modified PSO algorithms as an alternative for training an ANN [5] and [13]. However most of the research reported in these papers is focused only in the evolution of the synaptic weights, sometimes in the optimum selection of the number of neurons in hidden layers [14].

Due to the efficiency and adaptability of this type of techniques, in this paper we explore a new way of designing an ANN based on a modified version of PSO. This research includes not only the problem of finding the optimal set of synaptic weights of an ANN but also its topology and the transfer functions. In other words, given a set of inputs and desired patterns, with the proposal we will be able to find the best topology, the number of neurons, the transfer function for each neuron and the synaptic weights in order to design an ANN that can be used to solve a given problem. Finally, the proposal is tested in several non-linear problems.

II. Feed-forward neural networks

A neural network is a massively parallel-distributed processor made up from simple processing units, which has a natural propensity for storing experiential knowledge and making it available for use. The use of neural network offers the input-output mapping property and capability [6]. The artificial neuron, the most fundamental computational unit, is modeled based on the basic property of a biological neuron. This type of processing unit performs in two stages: weighted summation and some type of non-linear function. Each value of an input pattern $\mathbf{a} \in \mathbb{R}^N$ is associated with its synaptic weight values $\mathbf{W} \in \mathbb{R}^N$, which is normally between 0 and 1. Also, the summation function often takes an extra input value θ with weight value of 1 to represent

Manuscript received March 9, 2009. This work was economically supported by SIP-IPN under grants 20082948 and 20091421 and CONACYT under grant 46805.

Beatriz A. Garro is with the Center for Computer Research, National Polytechnic Institute CIC-IPN, Mexico City, DF 07738 MEXICO (e-mail: bgarrol@ipn.mx).

Roberto A. Vazquez is with the Center for Computer Research, National Polytechnic Institute CIC-IPN, Mexico City, DF 07738 MEXICO (e-mail: ravem@ipn.mx).

Humberto Sossa is with the Center for Computer Research, National Polytechnic Institute CIC-IPN, Mexico City, DF 07738 MEXICO (e-mail: hsossa@cic.ipn.mx).

978-1-4244-3553-1/09/$25.00 ©2009 IEEE

threshold or *bias* of a neuron. The summation function will be then performed as

$$o = \sum_{i=1}^{N} a_i w_i + \theta . \quad (1)$$

The sum-of-product value is then passed into the second stage to perform the activation function $f(o)$ which generates the output from the neuron and determines the behavior of the neural model. Among the most popular activation functions we could mention the logistic sigmoid.

By connecting multiple neurons, the true computing power of the neural networks emerges. The most common structure of connecting neurons into a network is by layers. In a multilayer structure, the input nodes, which received the pattern $\mathbf{a} \in \mathbb{R}^N$, pass the information to 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 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 a neural network are adapted through a continuing process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place.

In a general sense, the learning process may be classified as follows: supervised learning or unsupervised learning. Supervised learning assumes the availability of a labeled set of training data made up of p input-output samples:

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

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

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

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

One of the most commonly used supervised ANN model is back-propagation network that uses back-propagation (BP) algorithm [7-8] to minimize the objective function described in equation 3. The BP algorithm employs gradient descent by following the slope of Root Mean Square (RMS) error value along with the change in all the weight values. The weight values are constantly adjusted until the value of the error is no longer decreasing. Since the RMS error value is very a complex function with many parameter values of weights, it is possible that the back-propagation network may converge into a *local minimum* instead of the desired global minimum. This phenomenon can be avoided with several solutions, for example by presenting training samples to the learning network, adding noise to the weights while being updated or utilizing *momentum*, which gradually increases the weight adjustment rate. All of these solutions are the way to escape from the trap of a local minimum.

III. Particle Swarm Optimization

The particle swarm optimization (PSO) algorithm is a method for the optimization of continuous non-linear functions proposed by James Kennedy and Russell C. Eberhart [9]. This algorithm is inspired on observations of social and collective behavior as well as fish schooling or bird flocking and the model human social behavior.

PSO algorithm is a metaphor of the social behavior. For the case of a human, this social behavior is inspired in the fact that when a human being takes a decision, this decision is based on his own experience and in the experience of successful humans. For the case of a bird flocking, the social behavior is inspired on the movement of the flock in the search of food. Particularly, the behavior of a bird in the flock is based on the movements of the best member and at the same time also on his own experience.

For example, a population or a flock could be considerer like a cumulus of particles i where each particle represents the position $\mathbf{x}_i$ of a particle in a multidimensional space. These particles 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 as

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

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 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 as:

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

Finally, this optimum position will be the solution which maximize or minimize an objective function.

A. Basic PSO Algorithm

There are several versions of PSO algorithms which try to solve problems in different domains. In general, the PSO algorithm has three versions: a binary, real and hybrid version. The algorithm for the real version could be performed as follow:

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 criteria is reached:
   a) For each individual $\mathbf{x}_i$ evaluates 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 $\mathbf{x}_i(t+1)$.

B. Improved PSO

Shi and Eberhart [17] proposed a linearly varying inertia weight ω over the course of generations, which significantly improves the performance of PSO.

$$\omega = (\omega_1 - \omega_2) \times \frac{MAXITER - iter}{MAXITER} + \omega_2 \quad (6)$$

where ω_1 and ω_2 are the initial and final values of the inertia weight, respectively, $iter$ is the current iteration number and $MAXITER$ is the maximum number of allowable iterations. The empirical studies in [17] indicated that the optimal solution can be improved by varying the value of ω from 0.9 at the beginning of the evolutionary process to 0.4 at the end of the evolutionary process for most problems.

Ratnaweera et al. [18] proposed a version of PSO based on time-varying acceleration coefficients, which reduce the cognitive component and increases the social component by changing the acceleration coefficients c_1 and c_2 with time. It encourages a large cognitive component and small social component at the beginning of the search to guarantee particles' moving around the search space and to avoid particles' moving toward the population best position. On the other hand, a small cognitive component and a large social component allow the particles to converge to the global optima in the latter of the search. The varying scheme of c_1 and c_2 is given as follows:

$$\begin{aligned} c_1 &= (c_{1f} - c_{1i}) \times \frac{iter}{MAXITER} + c_{1i} \\ c_2 &= (c_{2f} - c_{2i}) \times \frac{iter}{MAXITER} + c_{2i} \end{aligned} \quad (7)$$

where c_{1f}, c_{1i}, c_{2f} and c_{2i} are constants, is the maximum number of allowable iterations. Through empirical studies, Ratnaweera et al. [18] have observed that the optimal solutions on most of the benchmarks can be improved by changing c_1 from 2.5 to 0.5 and changing c_2 from 0.5 to 2.5 over the full range of the search.

J. Wu et al. [14] developed an strategy that when the global best position is not improving with the increasing number of generations, each particle i will be selected by a predefined probability from the population, and then a random perturbation is added to each dimension of the velocity vector $\mathbf{v}_i$ of the selected particle i. The velocity resetting is presented as follows:

$$\mathbf{v}_i = \mathbf{v}_i + (2 \times r - 1) \times v_{\max} \quad (8)$$

where r is a uniformly distributed random numbers in the range $(0,1)$, and $v_{\max}$ is the maximum magnitude of the random perturbation to each dimension of the selected particle.

Based on some evolutionary schemes of GA, several effective mutation and crossover operators have been proposed for PSO. Lovbjerg et al. [19] proposed a crossover operator, in terms of a certain crossover rate α

$$ch_1(\mathbf{x}_i) = r_i par_1(\mathbf{x}_i) + (1 - r_i) par_2(\mathbf{x}_i) \quad (9)$$

where r_i is a uniformly distributed random numbers in the range $(0,1)$, ch_1 is the offspring and par_1 and par_2 are the two parents randomly selected from the population.

The velocity of the offspring is calculated as the sum of the velocity vectors of the two parents normalized to the original length of each parent velocity vector

$$ch_1(\mathbf{v}_i) = \frac{par_1(\mathbf{v}_i) + par_2(\mathbf{v}_i)}{\left| par_1(\mathbf{v}_i) + par_2(\mathbf{v}_i) \right|} \left| par_1(\mathbf{v}_i) \right| \quad (10)$$

Higashi and Iba [20] proposed a Gaussian mutation operator to improve the performance of PSO in term of a certain mutation rate β

$$ch(\mathbf{x}_i) = par(\mathbf{x}_i) + \frac{MAXITER - iter}{MAXITER} N(0,\sigma) \quad (11)$$

where ch is the offspring, par is the parent randomly selected from the population, $iter$ is the current iteration number and $MAXITER$ is the maximum number of allowable iterations.

Utilization of these operators in PSO have potential to achieve faster convergence and to find better solutions.

Mohais et al. [12, 13] used random neighborhoods in PSO, together with dynamism operators.

In this research we propose to use dynamic random neighborhoods that change in term of certain rate γ. First of all, a maximum number of neighborhoods $MAXNEIGH$ is defined in terms of the size of the population divided by 4. With this condition we assure that at least each neighborhood will have 4 members. Then the members of each neighborhood k are randomly selected and the best particle of each neighborhood $\mathbf{p}_{g_k}$ is computed. Finally the velocity of each particle i is updated as

$$\mathbf{v}_i(t+1) = \omega \mathbf{v}_i(t) + c_1 r_1 (\mathbf{p}_i(t) - \mathbf{x}_i(t)) + c_2 r_2 (\mathbf{p}_{g_k}(t) - \mathbf{x}_i(t)) \quad (12)$$

for all $i \in k, k = 1,\ldots,MAXNEIGH$.

In this research, the varying schemes of inertia weight ω and acceleration coefficients c_1 and c_2, velocity resetting, crossover and mutation operators, and dynamic random neighborhoods have been used in the original version of PSO in order to design an ANN given a set of patterns $\mathbf{T}$.

The algorithm for doing this task 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 criteria is reached:
   a) If $\mathbf{p}_g$ is not improved during $NITER$ perform a velocity resting.
   b) If $r(0,1) > \gamma$ then create new neighborhoods.
   c) Modify inertia weight.
   d) For each individual $\mathbf{x}_i$ evaluates their fitness.
   e) For each individual i, update its best position $\mathbf{p}_i$.
   f) For each neighborhood k, update the best individual $\mathbf{p}_{g_k}$.
   g) For each individual i and each dimension:
      i) compute the velocity update equation $\mathbf{v}_i(t+1)$.
      ii) Compute the current position $\mathbf{x}_i(t+1)$.
      iii) If $r(0,1) > \alpha$ apply crossover operator.
      iv) If $r(0,1) > \beta$ apply mutation operator.

IV. Design of an ANN using PSO

In this section it is described how given a set of patterns $\mathbf{T}$, an ANN can be automatically designed by means of the modified version of PSO already proposed. This proposal evolves the architecture of the ANN which includes its topological structure and transfer function of each neuron which have a significant impact on the ANN's performance. Nowadays, architecture design depends on the human experience and a trial and error process, for that reason, evolving an ANN to find the optimal architecture is suitable.

In this research the maximum number of neurons is defined as follows

$$MNN = (nfin + nfout) \times 2 \quad (13)$$

where $nfin$ is the dimension of the input patterns vector and $nfout$ is the dimension of the desired patterns vector.

A. Coding scheme

The main problem when ANNs are designed involves the selection of the best procedures to obtain an optimal ANN structure. Our proposal, simultaneously, evolves the topology, synaptic weights and the transfer function for each neuron. In order to achieve this goal, we codify this information as follows: each particle will be represented by a matrix $\mathbf{x} \in \mathbb{R}^{(MNN+2)\times MNN}$ composed by three parts: topology, synaptic weights and transfer function.

The topology of the ANN is codified based on the binary square matrix representation of a graph $\mathbf{x}$ where each component x_{ij} represents the connections between neuron i and neuron j when $x_{ij} = 1$. However, instead of evolving this binary information we decided to codify this information into its decimal base value and then evolve it. For example suppose that next binary code "01101" represents the connections of a *i*-th neuron to 5 neurons where only the neurons 2, 3, and 5 are connected to *i*. This binary code is transformed into its decimal base value resulting in "13", which will be the number that we will evolve instead of the binary value. This scheme is much faster. Instead of evolving a string of MNN bits, we evolve only a decimal base number.

The synapses weights of the ANN are codified based on the square matrix representation of a graph $\mathbf{x}$ where each

component x_{ij} represents the synaptic weight between neuron i and neuron j.

Finally, the transfer function for each neuron will be represented by an integer from 0 to 5 representing one of the 6 transfer functions used in this research: logsig, tansig, sin, radbas, pureline and hardlim. These functions were selected because they are the most popular and useful transfer functions in several kinds of problems.

Figure 1 shows the individual representation of the ANN.

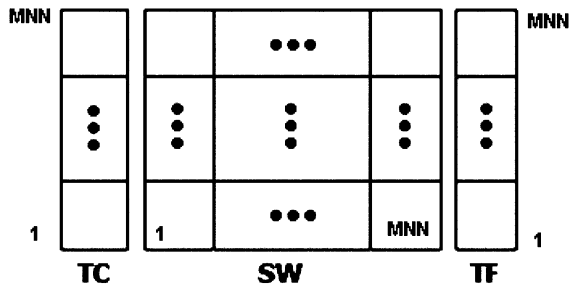


Fig. 1. Individual representation of an ANN. Note that TC represents the topology of the network, SW represents the synaptic weights and TF represents TF of each neuron.

Another alternative to codify the topology could be combined it with the synaptic weights where values less than a given threshold means no connection. However, determining the best threshold could bring other kind of problems; that is why in this paper we will not focus in this scheme.

B. Fitness Function

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 as follows:

1) For the first $nfin$ neurons, the output $o_i = a_i$.
2) For each neuron $n_i, i = nfin, \ldots, MNN$.
   a) Get connections by using individual $\mathbf{x}_{1,i}$.
   b) For each neuron $j < i$ connected to n_i, $o_i = f(o)$ where f is a transfer function given by individual $\mathbf{x}_{nfout,i}$ and o is compute using eq. 1.
3) Finally, $y_i = o_i, i = MNN - nfouy, \ldots, MNN$.

Note that the restriction $j < i$ is used to avoid the generation of cycles in the ANN.

V. Experimental Results

In order to evaluate the accuracy of the already proposed modified PSO algorithm, several experiments were performed using 4 data sets. Three of them were taken from UCI machine learning benchmark repository [21]: XOR, iris plant, wine, and breast cancer.

All data sets were partitioned into two sets: a training set and a testing set. For the iris plant data set, the first 120 examples were used for the training set, and the remaining 30 examples for the testing set. For the wine set, the first 90 examples were used for training set, and the remaining 89 examples for the testing set. For the breast cancer, the first 600 examples were used for training set, and the remaining 83 for testing set. For the XOR problem, the first 4 examples of the 2-dimensional version were used for training, and noisy versions of these examples were used for the testing set.

For the iris plant data set, the dimension of input vector is 4 and the number of classes is 3. For the wine data set, the dimension of input vector is 13 and the number of classes is 3. For the breast cancer data set, the dimension of input vector is 9 and the number of classes is 2.

The input features of all data set were rescaled in a range between $[0,1]$. The outputs were encoded by a binary representation of the class to which input patterns belongs. For the wine data set, the output was encoded by the 1-to- c representations for c classes.

Before starting with the experiments, we have to define the parameters of the proposed PSO algorithm. These parameters were set to the same value for all the data set problems: population size M equal to 50, number of generations equal to 2000, $NITER$ equal to 10, initial position of the particles between $[-4,4]$, velocity range $[-2,2]$, inertia weight from 0.9 to 0.4, $\gamma = 0.05$, $\alpha = 0.4$ and $\beta = 0.6$.

For the XOR data set the proposed PSO algorithm designed 4 ANN, the error of the proposal achieved during training and testing phase was of 0 for the 4 ANN. Fig. 2 shows the 4 ANN designed by the proposed PSO algorithm. It is important to notice that the topology and the transfer functions for each neuron were automatically determined by the proposed PSO algorithm. Note that different to traditional feed-forward topologies, the topologies already obtained have lateral connections and connections between input and output layers.

Numbering the neurons of the networks from top-down and left-right and numbering the transfer function as *logsig* (0), *tansig* (1), *sin* (2), *radbas* (3), *pureline* (4) and *hardlim* (5). The following notation was defined: $\langle nn, tfn \rangle$ where nn represents the neuron's number and tfn represent a transfer function number; for example $\langle 3,1 \rangle$ indicates that neuron 3 uses the *radbas* transfer function.

The ANN shown in Fig. 2(a) is defined as follows: $\langle 3,4 \rangle$, $\langle 4,5 \rangle$, $\langle 5,4 \rangle$ and $\langle 6,5 \rangle$. Fig. 2(b) is defined as follows:

$\langle 3,3\rangle$, $\langle 4,5\rangle$, $\langle 5,1\rangle$ and $\langle 6,5\rangle$. Fig. 2(c) is defined as follows: $\langle 3,5\rangle$, $\langle 4,1\rangle$, $\langle 5,1\rangle$ and $\langle 6,5\rangle$. Finally, Fig. 2(d) is defined as follows: $\langle 3,3\rangle$, $\langle 4,2\rangle$, $\langle 5,3\rangle$ and $\langle 6,2\rangle$.

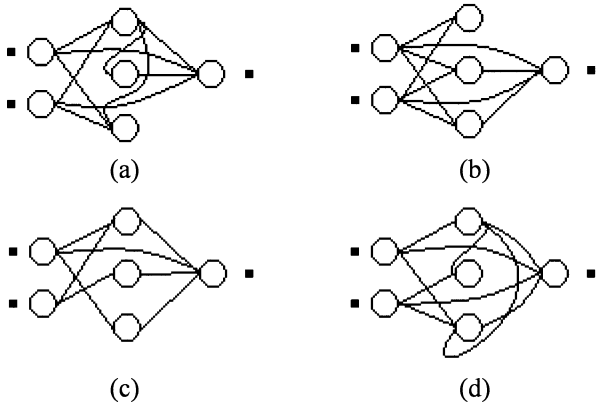


Fig. 2. Different topologies obtained using the proposed PSO algorithm for the XOR problem.

As you can appreciate the ANNs generated by the proposed PSO algorithm use different transfer functions and the connections between neurons are completely different.

For the iris plant data set the proposed PSO algorithm designed four ANN, the error of the proposal achieved during training phase was of 0.088, 0.099, 0.092 and 0.086, respectively. The error achieved during testing phase was of 0.064, 0.071, 0.057 and 0.073, respectively. Fig. 3 shows one of the 4 ANN designed by the proposed PSO algorithm.

The ANN shown in Fig. 3 is defined as follows: $\langle 5,3\rangle$, $\langle 6,3\rangle$, $\langle 7,2\rangle$, $\langle 8,3\rangle$, $\langle 9,3\rangle$, $\langle 10,2\rangle$, $\langle 11,3\rangle$, $\langle 12,3\rangle$, $\langle 13,2\rangle$ and $\langle 14,3\rangle$.

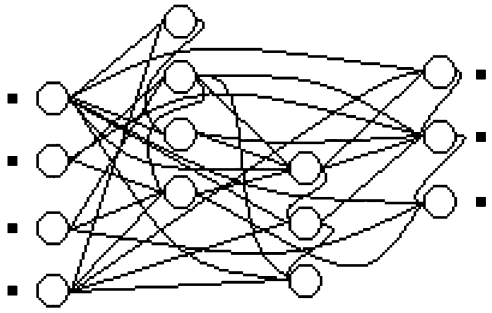


Fig. 3. One of the topologies obtained using the proposed PSO algorithm for the iris plant data set.

For the wine data set the proposed PSO algorithm designed an ANN, the error of the proposal achieved during training and testing phases was of 0.058 and 0.076, respectively. Fig. 4 shows the ANN designed by the proposed PSO algorithm.

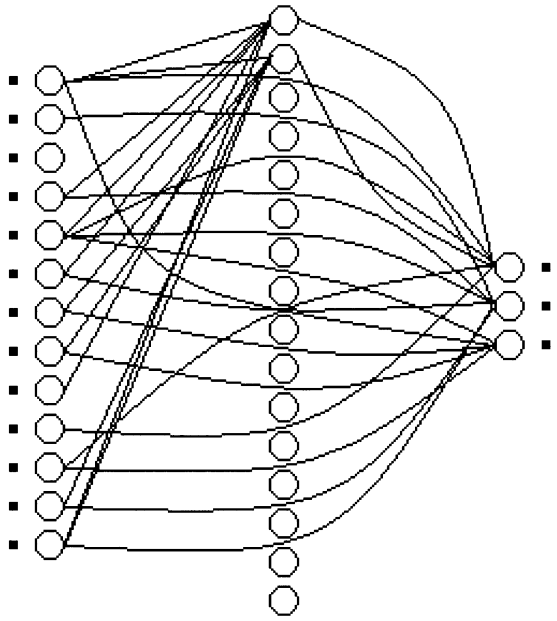


Fig. 4. Topology obtained using the proposed PSO algorithm for the wine data set.

The ANN shown in Fig. 4 is defined as follows: $\langle 14,3\rangle$, $\langle 15,3\rangle$, $\langle 16,3\rangle$, $\langle 17,3\rangle$, $\langle 18,2\rangle$, $\langle 19,1\rangle$, $\langle 20,3\rangle$, $\langle 21,1\rangle$, $\langle 22,1\rangle$, $\langle 23,3\rangle$, $\langle 24,1\rangle$, $\langle 25,2\rangle$, $\langle 26,1\rangle$, $\langle 27,1\rangle$, $\langle 28,2\rangle$, $\langle 29,2\rangle$, $\langle 30,3\rangle$, $\langle 31,4\rangle$ and $\langle 32,4\rangle$. It is important to notice that not all the neurons contribute to the output of the ANN, in other words, the use of several neurons not always is reflected in a better result.

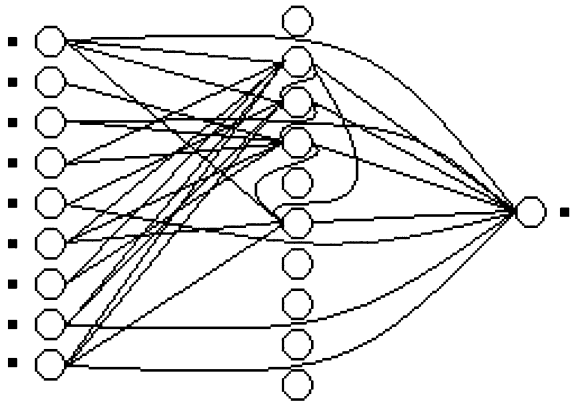
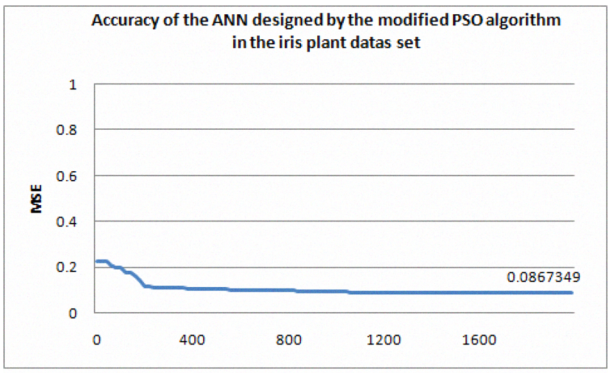


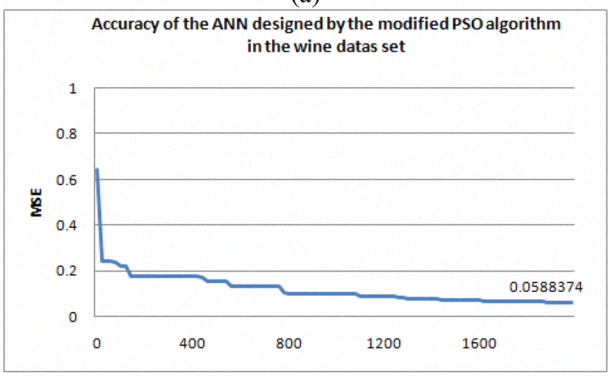
Fig. 5. Topology obtained using the proposed PSO algorithm for the breast cancer data set.

Finally, for the breast cancer data set the proposed PSO algorithm designed an ANN, the error of the proposal achieved during training and testing phases was of 0.013 and 0.035, respectively. Fig. 5 shows the ANN designed by the proposed PSO algorithm. The ANN shown in Fig. 5 is

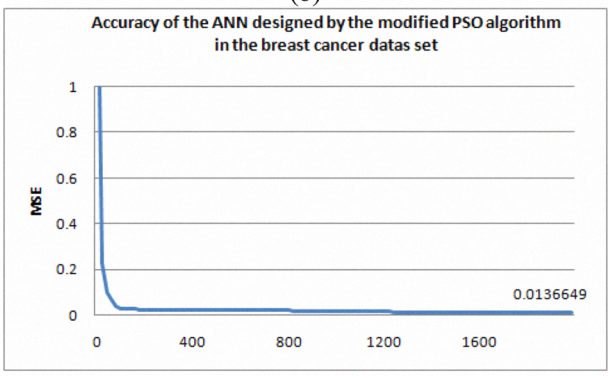
defined as follows: $\langle 10,3\rangle$, $\langle 11,3\rangle$, $\langle 12,1\rangle$, $\langle 13,1\rangle$, $\langle 14,1\rangle$, $\langle 15,1\rangle$, $\langle 16,2\rangle$, $\langle 17,2\rangle$, $\langle 18,2\rangle$, $\langle 19,3\rangle$ and $\langle 20,3\rangle$.



(a)



(b)



(c)

Fig. 6. Accuracy of the ANNs designed by the modified PSO algorithm. (a) Iris plant data set. (b) Wine data set. (c) Breast cancer data set.

Based on the previous results we can consider PSO algorithms as an alternative to automatically design an ANN using only a label set of training data. Compared against back-propagation (BP) algorithm for an ANN composed by 3 layers, learning rate was of 0.1 for the same data sets and the same number of epochs, PSO is better. For the XOR problem, during training and testing phases, the error achieved by the ANN (2-2-1) was of 0.0097 and 0.011, respectively. For the iris data set during training and testing phases, error achieved by the ANN (4-6-3) was of 0.0075 and 0.128, respectively; although in this case training error was better than the error obtained by our proposal, the testing error achieved by BP algorithm was worse. This fact suggests that our proposal is better because generalization is the most important task of an ANN.

For the wine data set during training and testing phases, error achieved by the ANN (13-5-3) was of 0.00011 and 0.016, respectively. Finally, for the breast cancer data set during training and testing phases, error achieved by the ANN (9-5-1) was of 0.0085 and 0.013, respectively. In this case, it seems that results obtained by BP algorithm are a little bit better than the results obtained by our proposal. It is important to note that our proposal designed automatically the neural network without the intervention of an expert, and in this particular case, the results are comparable to those obtained by BP algorithm.

During all the experimentation, it was observed that the most used transfer functions by the neurons in the 4 benchmarks were the *logsig* and *sin* functions.

In Fig. 6 it is shown the evolution of the error achieved by the ANNs designed by modified PSO in the iris plant, wine and breast cancer data sets.

In addition, Table I, it is shown an accuracy comparison among two PSO algorithms and our proposal. As you can appreciate, under the same conditions and configuration, our proposal was better. While basic PSO and second generation PSO [24] needed more than 2000 epochs to converge in good results, our proposal only needed 2000 epochs.

TABLE I
ACCURACY COMPARISON OF TWO PSO ALGORITHMS AGAINST THE PROPOSED PSO ALGORITHM

Data base	Basic PSO		Second generation PSO		Proposed PSO Algorithm	
	Tr. Er.	Te. Er.	Tr. Er.	Te. Er.	Tr. Er.	Te. Er.
XOR	0	0	0	0	0	0
Iris plant	0.202	0.237	0.184	0.173	0.092	0.057
Wine	0.266	0.305	0.276	0.295	0.058	0.076
Breast Cancer	0.032	0.018	0.325	0.316	0.013	0.035

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

VI. CONCLUSIONS

In this paper an alternative way to automatically design an ANN was proposed. This new alternative is based on a bio-inspired technique called Particle Swarm Optimization (PSO). Due to the design of a neural network can be seen as an optimization problem, which consists on finding the best values of the synapses which minimize an error function, the best topology and the best transfer functions for each neuron, PSO algorithm is suitable to find all of this information and optimize the error function (minimize an objective function).

The accuracy of the proposal was tested using several non-linear problems and the results show a clear advantage against traditional schemes which involve the selection of a learning algorithm, a topology and the transfer functions. As a result of this research we proposed a modified PSO

algorithm capable to automatically design ANNs only taking into account a label set of training data.

Compared against back-propagation (BP) algorithm in ANN composed by 3 layers, learning rate of 0.1, the same data sets and the same number of epochs, our proposed PSO algorithm was better. Furthermore our proposal seems to need fewer epochs to get good results compared against basic PSO and second generation PSO algorithms.

Nowadays we are investigating how much the behavior of the proposed PSO algorithm changes when the parameters are tuned. One of the future improvements of this PSO algorithm could be reached by combining PSO another evolutionary strategy such as differential evolution. Furthermore, we are testing this approach in the design of ANNs in segmentation tasks such for example in face segmentation and road segmentation in images.

REFERENCES

[1] S. Tejen, J. Jyunwei and H. Chengchih, "A Hybrid Artificial Neural Networks and Particle Swarm Optimization for Function Approximation," *International Journal of Innovative Computing, Information and Control ICIC*, vol. 4, pp. 2363-2374, 2008.

[2] K.W. Chau, "Application of a PSO-based neural network in analysis of outcomes of construction claims," *Automation in Construction*, vol.16, pp. 642-646, 2007.

[3] A. Chatterjee, K. Pulasinghe, K. Watanabe and K. Izumi, "A Particle Swarm Optimized Fuzzy-Neural Network for Voice Controlled Robot Systems," *IEEE Transactions on Industrial Electronics*, vol. 52, pp. 1478 – 1489, 2005.

[4] Z. Wang, G. L. Durst, R. C. Eberhart, D. B. Boyd and Z. Miled, "Particle Swarm Optimization and Neural Network Application for QSAR," *Proceedings 18th Parallel and Distributed Processing Symposium*, 2004.

[5] L. Zhao and Y. Yang, "PSO-Based Single Multiplicative Neuron Model for Time Series Prediction," *Expert Systems with Applications*, vol.36, pp.2805-2812, 2009.

[6] D. E Rumelhart, G. E. Hinton and R. J.Williams, "*Learning Internal Representations and Error Propagation*," in D. E. Rumelhart and J. L. McCleland, eds. (Cambridge, MA: MIT Press), Vol. 1, Chapter 8, 1986.

[7] J. A. Anderson, *"Introduction to Neural Networks,"* (Cambridge, MA: MIT Press), 1995.

[8] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proc. IEEE*, vol. 78, pp. 1550–1560, 1990.

[9] J. Kennedy and R. C. Eberhart with Yuhui Shi, "*Swarm Intelligence,"* Morgan Kaufmann Publishers, 2001.

[10] J. Kennedy, "Stereotyping: Improving particle swarm performance with cluster analysis," *In Proc. of the IEEE Congress on Evolutionary Computation*, pp. 1507–1512, 2000.

[11] M. Richards and D. Ventura, "Dynamic Sociometry and Population Size in Particle Swarm Optimization," *Proc. of the Sixth International Conference on Computational Intelligence and Natural Computing*, pp. 1557–1560, 2003.

[12] A. S. Mohais, R. Mohais, C. Ward and C. Posthoff, "Earthquake Classifying Neural Networks Trained with Random Dynamic Neighborhood PSOs," *GECCO*, pp. 110-117, 2007.

[13] Y. Da, X. R. Ge, "An improved PSO–based ANN with simulated annealing technique," *Neurocomput Lett*, vol. 63, pp. 527–533, 2005.

[14] J. Yu, L. Xiand S. Wang, "An Improved Particle Swarm Optimization for Evolving Feedforward Artificial Neural Networks," *Neural Processing Letters*, vol. 26, pp. 217-231, 2007.

[15] J. M. Yang and C. Y. Kao, "A robust evolutionary algorithm for training neural networks," *Neural Comput Appl*, vol. 10, pp. 214–230, 2001.

[16] P. A. Castillo, J. Carpio, J. J. Merelo, A. Prieto and V. Rivas, "Evolving multilayer perceptrons," *Neural Process Lett*, vol.12, pp. 115–127, 2000.

[17] Y. Shi and R. C. Eberhart, "Empirical study of particle swarm Opimization," *In: Proc. IEEE Int. Conf. Evolutionary Computation,* vol. 3, pp. 101–106, 1999.

[18] A. Ratnaweera, K. Saman and H. C. Watson, "Self–organizing hierarchical particle swarm optimizer with time–varing acceleration coefficients," *IEEE Trans Evol Comput*, vol. 8, pp. 240–255, 2004.

[19] M. Lovbjerg, T. K. Rasmussen and T. Krink, "Hybrid particle swarm optimiser with breeding and subpopulations," *In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO).* San Francisco, CA, July 2001

[20] N. Higashi and H. Iba, "Particle swarm optimization with Gaussian mutation," *In: Proc. of the IEEE Swarm Intelligence Symp.* Indianapolis: IEEE Inc pp. 72–79, 2003.

[21] P. M. Murphy and D. W. Aha, *"UCI repository of machine learning databases,"* Dept. Inf. Comput. Sci., Univ. California, Irvine, CA, 1994.

[22] D. Rivero, J. Rubañal, J. Dorado and A. Pazos, "Automatic design of ANN by means of GP for data mining tasks: iris flower classification," *Lecture Notes in Computer Sciences*, 4431, pp. 276-285, 2007.

[23] X. Yao, "Evolving Artificial Neural Networks," *Proceedings of IEEE*, vol. 87, no. 9, pp. 1423-1446,1999.

[24] M. Chen, "Second generation particle swarm optimization," *IEEE Congress on Evolutionary Computation, CEC 2008*, pp. 90-96, 2008