Back-Propagation vs Particle Swarm Optimization Algorithm: which Algorithm is better to adjust the Synaptic Weights of a Feed-Forward ANN?

Beatriz A. Garro¹, Humberto Sossa¹ and Roberto A. Vázquez²
¹ Centro de Investigación en Computación – IPN
Av. Juan de Dios Batiz, esquina con Miguel de Othon de Mendizábal
Ciudad de México, 07738, México
²Facultad de Ingeniería – Universidad La Salle
Benjamín Franklin 47 Col. Condesa CP 06140 México, D.F
Contact: bgarrol@ipn.mx, hsossa@cic.ipn.mx, ravem@lasallistas.org.mx

Abstract
Bio-inspired algorithms have shown their usefulness in different non-linear optimization problems. Due to their efficiency and adaptability, these algorithms have been applied to a wide range of problems. In this paper we compare two ways of training an artificial neural network (ANN): Particle Swarm Optimization (PSO) algorithms against classical training algorithms such as: back-propagation (BP) and Levenberg Marquardt method. The main contribution of this paper is to answer the next question: is PSO really better than classical training algorithms in adjusting the synaptic weights of an ANN? First of all, we explain how the ANN training phase could be seen as an optimization problem. Then, it is explained how PSO could be applied to find the best synaptic weights of the ANN. Finally, we perform a comparison among different classical methods and PSO approach when an ANN is applied to different non-linear problems and to a real object recognition problem.

Keywords: Artificial neural networks, particle swarm intelligence, pattern recognition
Computing Classification System (CCS): I.2.8, I.5.1

1. INTRODUCTION

Particle swarm optimization (PSO) algorithm is inspired by observations of social interaction. 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 search an optimum solution based on the self experience and social experience of the best particle of the population.

A feed-forward artificial neural network (ANN) is a powerful tool widely used in the field of pattern recognition and time series forecasting. However, despite of their power in some practical problems, neural networks can not reach an optimum performance in several non-linear problems. This is mainly due to the parameters used during learning phase: learning rate, momentums, and so on. In general, these parameters do not allow compute the best set of synaptic weights.

Several works evolutionary-based-strategies for training ANN have been reported in the literature. Evolutionary algorithms (EAs) are non-gradient approaches. They have proven to be promising for training ANNs. For example, in (Tejen et al., 2008), the authors combine PSO and ANNs for function approximation. Other application presented in (Chau, 2007) is a PSO-based ANN for construction claims analysis in Hong Kong. In (Chatterjee, 2005), authors use PSO in fuzzy-neural networks for voice-controlled robot systems. In (Wang, 2004), a PSO technique is used to select the most important characteristics from a set of patterns. In general, the authors present modified PSO algorithms as an alternative for training an ANN (Zhao and Yang, 2009) (Da and Ge, 2005), however most of the
research is focused only in the evolution of the synaptic weights and sometimes in the optimum selection of the number of neurons in hidden layers (Yu et al., 2007). In (Gudise and Venayagamoorthy, 2003) the authors present a comparative between PSO and back-propagation (BP) algorithms for training feed-forward neural networks applied to solve a quadratic function (regression problem) in order to demonstrate the capacities of the algorithms. In (Mohaghegi et al., 2005) the authors compare the same algorithms in order to train a radial base network function. However, the authors do not analyze the behavior and robustness of PSO in some other kind of problems.

Other types of EAs as genetic algorithms and genetic programming have been applied to the same problem; refer for example to (Yang and Kao, 2001), (Castillo et al., 2000) and (Rivero et al., 2007).

In this paper we compare the accuracy of three algorithms (PSO, BP and Levenberg Marquardt’s method). These algorithms are used to train the synaptic weights of an ANN applied to solve different classification problems such as: iris plant database, wine database and a real object recognition problem. All of these experiments were performed with the aim to answer the next question: does really this kind of algorithms help to improve the accuracy of an ANN? In particular, we investigate if PSO is better than some classical training algorithms in the task of adjusting the synaptic weights of an ANN. It is important to remark that we compare these three strategies only by adjusting the synaptic weights. This way the capabilities of an algorithm are better understood when the problem is analyzed in its simplest way.

First of all, we explain how the neural network training phase could be seen as an optimization problem. Then, we explain how PSO could be applied to find the best synaptic weights of an ANN. Finally, we perform a comparison between some classical methods and PSO algorithms for adjusting the ANN’s weights to solve different non-linear pattern recognition problems and a real object recognition problem.

It is worth mentioning that a first step in this direction was described in (Garro et al., 2010).

2. BASICS ON FEED-FORWARD NEURAL NETWORKS

A neural network is 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 non-linear function. It accepts a set of inputs to generate the weighted sum then passes the result to the non-linear function to make an output.

Each value of an input pattern \(A \in \mathbb{R}^N\) is associated with its weight value \(W \in \mathbb{R}^N\), which is normally between 0 and 1. Also, the summation function often takes an extra input value \(\theta\) with weight value of 1 to represent threshold or bias of a neuron. The summation function will be then performed as,

\[
y = \sum_{i=1}^{N} a_i w_i + \theta
\]

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.
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 \( x \in \mathbb{R}^N \), 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 \( y \in \mathbb{R}^M \).

Basically, learning is a process by which the free parameters (i.e., synaptic weights \( W \) and bias levels \( \theta \)) of a neural network are adapted through a continuing process of stimulation by the environment in which the network is embedded. 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:

\[
T^\xi = \{(x^\xi \in \mathbb{R}^N, d^\xi \in \mathbb{R}^M)\} \forall \xi = 1, \ldots, p
\]

where \( x \) is the input pattern and \( d \) the desired response.

Given the training sample \( T^\xi \), the requirement is to compute the free parameters of the neural network so that the actual output \( y^\xi \) of the neural network due to \( x^\xi \) is close enough to \( d^\xi \) for all \( \xi \) 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:

\[
e = \frac{1}{p \cdot M} \sum_{\xi=1}^{p} \sum_{i=1}^{M} (d_i^\xi - y_i^\xi)^2
\]

One of the most commonly used supervised ANN model is feed-forward network that uses backpropagation (BP) learning algorithm [7-8] to minimize the objective function described in eq. 3.

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 BP network may converge into a local minimum instead of the desired global minimum. This phenomenon can be avoided with several solutions for example 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.

3. BASICS ON PARTICLE SWARM OPTIMIZATION

Particle swarm optimization (PSO) algorithm is a method for the optimization of continuous non-linear functions proposed by James Kennedy and Russell C. Eberhart. It is inspired on observations of social and collective behavior as well as fish schooling or bird flocking and the model human social behavior (Kennedy et al., 2001).
PSO algorithm is a metaphor of the social behavior. 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 considered like as a cumulus of particles \( i \) where each particle represents the position \( x_i \) of a particle in a multidimensional space. These particles also represent a possible solution of a specific optimization function. According to a velocity function \( v_i \) which takes into a count the best position of a particle in a population \( p_g \) (i.e. social component) as well as the own best position of the particle \( p_i \) (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 the following equation:

\[
v_i(t+1) = \omega v_i(t) + c_1 r_1 (p_i(t) - x_i(t)) + c_2 r_2 (p_g(t) - x_i(t))
\]  

(4)

where \( \omega \) 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 \( v_i \) 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 \( p_i \), and the global best position \( p_g \). Based on the updated velocities, the new position of the particle \( i \) is computed using

\[
x_i(t+1) = x_i(t) + v_i(t+1).
\]  

(5)

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

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, a real and a hybrid version. The algorithm for the real version could be performed as follows:

Given a population of \( x_i \in \mathbb{R}^D, i = 1, \ldots, M \) individuals

1) Initialize the population at random
2) Until a stop criteria is reached:
   a) For each individual \( x_i \), evaluate their fitness.
   b) For each individual \( i \), update its best position \( p_i \).
   c) From all individual \( i \), update the best individual \( p_g \).
d) For each individual $i$, compute the velocity update equation $v_i(t+1)$ and then compute the current position $x_i(t+1)$.

4. EVOLVING THE SYNAPTIC WEIGHTS OF AN ANN USING PSO

In this section it is described how given a set of patterns $T$, the synaptic weights of an ANN can be automatically adjusted by means of a basic PSO. It is important to remark that the architecture of the ANN has to be previously defined.

Each particle (individual) of the population is represented by means of a vector that will be evolved by PSO algorithm. This vector is a solution composed with the set of synaptic weights $w_{ji}^k$, between neuron $i$ and neuron $j$ that belongs to the layer $k$, which maximize the performance of the ANN, refer to Fig. 1. This particle will change at each iteration; it will be evaluated in terms of a fitness function in order to calculate the minimum square error generated by the ANN. The particle whose weights provoke the minimum value of the MSE will be the best solution for the trained ANN.

Finally, the fitness function which measures the performance of each individual is given by eq. 3 where the output $y_i$ of the ANN is computed by means of eq. 1.

5. EXPERIMENTAL RESULTS

In order to evaluate the accuracy of the PSO algorithm and compare it against the classical training algorithms, several experiments using three data sets were performed. Two of them were taken from UCI machine learning benchmark repository (Murphy and Aha, 1994): iris plant dataset and wine dataset. In addition we compare the accuracy of the algorithms when they are applied to a real object recognition problem.

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 training; the remaining 30 examples were used for testing. For the wine set, the first 90 examples were used for training, while the remaining 89 examples were used for testing.

For the iris plant data set, the number of input features is four and the number of classes is three. For the wine data set, the number of input features is 13 and the number of classes is three.

For the case of the real object recognition problem we used a set of 100 images which contains five different objects whose images are shown in Figure 2. Objects were not recognized directly from their images. We preferred to do it by an invariant description of each object. We used 10 images of each object in different positions, rotations and scale changes. To each image of each object a standard thresholder (Otsu, 1979) was applied to get its binary version. Small spurious regions were eliminated form each image by means of a size filter (Jain, 1995). To each of the 10 images of each object the seven well-known Hu's geometric invariants, to translations, rotations and scale changes, were computed (Hu, 1962). Finally we used this information to train the ANN. The remaining 50 images were used for testing the ANN following the same procedure to obtain the Hu invariants.
The input features of all data sets were rescaled in a range between $[0,1]$. The outputs were encoded by a binary representation of the class to which input patterns belongs.

Before starting with the experiments, we have to define 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 10000, the initial position of the particles between was in the range $[0,1]$. Velocity range $[-0.01,0.01]$, $\omega = 0.729$, $c_1 = c_2 = 2$. For the case of the classic BP algorithm and Levenberg Marquardt method the number of generations was set to 10000, $\alpha = 0.1$ and $\beta = 0.3$. These parameters were set to the same value for all the experiments. The stop criteria in both algorithms was the number of generations or a minimum error set to $10^{-6}$. We performed 5 runs for each data set.

Table 1 shows the experimental results for the iris plant data set. For this problem BP algorithm provided better results than PSO algorithm. Levenberg Marquardt method provides the worst results. In the three algorithms the training error was smaller than the testing error. Furthermore, neither of the algorithms reached the desired minimum training error.

Table 2 shows the experimental results for the wine data set. For this problem PSO algorithm provided better results than classic BP algorithm and Levenberg Marquardt method during training. However, during testing, BP algorithm provided similar results compared to those provided with PSO algorithm. Other important fact that we would like to remark is that PSO reached the minimum error goal in less epochs compared against BP algorithm and Levenberg Marquardt method.

Table 3 shows the experimental results for the real object recognition problem. For this problem PSO algorithm is much better than the BP algorithm during the training; also PSO algorithm provided better results than BP algorithm in the testing stage. In addition PSO found minimum error goal in much less epochs. However Levenberg Marquardt method provided the best result during training and the number of epochs that the method used to reach the goal error was less than the PSO algorithm required.

In addition, from Tables 4 and 5 you can appreciate the average and standard deviation error from the accuracy of all experimental results. The reader can observe that is difficult to decide which algorithm is better during the training and testing stages. In some cases PSO is better than BP, in other cases BP is better than PSO and sometimes Levenberg Marquardt method is better than PSO. However PSO algorithm is nearer to the minimum error and the number of epochs is less than the BP algorithm. This fact does not mean than PSO can substitute the BP algorithm or Levenberg Marquardt method for training the ANN, but could be consider as a new tool for training ANN. Nevertheless, for a real application PSO was better, but if we analyze the patterns extracted from the real objects, we observe that classes are linearly separable.

Although most of the times, classical training methods provide good solutions, there are several problems where the solution space is very complex due to non-differentiable spaces. In that case PSO algorithm could be more useful for training an ANN.

6. CONCLUSIONS

In this paper we tried to answer to the next question: a bio-inspired algorithm, particularly PSO is better than a classical training algorithm, such as back-propagation algorithm and Levenberg Marquardt
method in adjusting the synaptic weights of an ANN? This question was not completely solved because its answer depends a lot of the problem to be solved and also of the predefined architecture of the network.

It is important to remark that we compared these three approaches only in the problem of adjusting the synaptic weights. This was because in our opinion to understand the capabilities of an algorithm is better to analyze it when it is applied in a simple problem.

We explained in detail how the neural network training phase could be seen as an optimization problem. Then, we explained how PSO could be applied to find the optimal synaptic weights of the neural network. Finally, we performed a comparison between three classical algorithms and PSO algorithm in adjusting the ANN’s weights to solve different non-linear problems, and a real object recognition problem.

From these experiments we observed that PSO presents a better accuracy than classical algorithm in linear problems and for the cases of non-linear problem we could not distinguish a clear winner. However, as was shown in some other papers ((Garro et al., 2009) for example), the advantage of PSO against to BP algorithms is shown when PSO evolves other kind of parameters that BP algorithms cannot. Some examples of these parameters could be to find the best topology, the best transfer function and so on.

Acknowledgements. The authors thank SIP-IPN for the economical support under grant 20100468. 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. We thank also the reviewers for their comments for the improvement of this paper.

REFERENCES


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

Fig. 2. (a-e) Some of the images used to train the ANN. (f-j) Some of the images used to test the ANN.

Table 1. Experimental results obtained using the iris plant problem and an architecture 4-5-3 (4 neurons in input layers, 5 neurons in hidden layers and 3 neurons in output layers).

<table>
<thead>
<tr>
<th>Run</th>
<th>BP algorithm</th>
<th>Levenberg Marquardt method</th>
<th>PSO algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.073</td>
<td>0.112</td>
<td>10000</td>
</tr>
<tr>
<td>2</td>
<td>0.072</td>
<td>0.117</td>
<td>10000</td>
</tr>
<tr>
<td>3</td>
<td>0.076</td>
<td>0.102</td>
<td>10000</td>
</tr>
<tr>
<td>4</td>
<td>0.074</td>
<td>0.115</td>
<td>10000</td>
</tr>
<tr>
<td>5</td>
<td>0.076</td>
<td>0.113</td>
<td>10000</td>
</tr>
</tbody>
</table>

Tr. Er = Training Error, Te. Er. = Testing Error.
Table 2. Experimental results obtained using the wine problem and an architecture 13-4-3 (13 neurons in input layers, 4 neurons in hidden layers and 3 neurons in output layers).

<table>
<thead>
<tr>
<th>Run</th>
<th>BP algorithm</th>
<th>Levenberg Marquardt method</th>
<th>PSO algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.42E-5</td>
<td>0.018</td>
<td>10000</td>
</tr>
<tr>
<td>2</td>
<td>1.45E-5</td>
<td>0.025</td>
<td>10000</td>
</tr>
<tr>
<td>3</td>
<td>1.72E-5</td>
<td>0.017</td>
<td>1000</td>
</tr>
<tr>
<td>4</td>
<td>1.62E-5</td>
<td>0.023</td>
<td>10000</td>
</tr>
<tr>
<td>5</td>
<td>1.60E-5</td>
<td>0.023</td>
<td>10000</td>
</tr>
</tbody>
</table>

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

Table 3. Experimental results obtained using the real object recognition problem and an architecture 7-4-3 (7 neurons in input layers, 4 neurons in hidden layers and 3 neurons in output layers).

<table>
<thead>
<tr>
<th>Run</th>
<th>BP algorithm</th>
<th>Levenberg Marquardt method</th>
<th>PSO algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.75E-5</td>
<td>6.95E-5</td>
<td>10000</td>
</tr>
<tr>
<td>2</td>
<td>4.87E-5</td>
<td>6.64E-5</td>
<td>10000</td>
</tr>
<tr>
<td>3</td>
<td>4.67E-5</td>
<td>6.51E-5</td>
<td>10000</td>
</tr>
<tr>
<td>4</td>
<td>4.53E-5</td>
<td>6.42E-5</td>
<td>10000</td>
</tr>
<tr>
<td>5</td>
<td>4.78E-5</td>
<td>7.08E-5</td>
<td>10000</td>
</tr>
</tbody>
</table>

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

Table 4. Average experimental results obtained using the different problems.

<table>
<thead>
<tr>
<th>Data base</th>
<th>BP algorithm</th>
<th>Levenberg Marquardt method</th>
<th>PSO algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris plant</td>
<td>0.074</td>
<td>0.112</td>
<td>10000</td>
</tr>
<tr>
<td>Wine</td>
<td>1.56E-5</td>
<td>0.021</td>
<td>8740</td>
</tr>
<tr>
<td>Object Recognition</td>
<td>4.72E-5</td>
<td>6.72E-5</td>
<td>10000</td>
</tr>
</tbody>
</table>

Tr. Er = Training Error, Te. Er. = Testing Error.
Table 5. Standard deviation from the experimental results obtained using the different problems.

<table>
<thead>
<tr>
<th>Data base</th>
<th>BP algorithm</th>
<th>Levenberg Marquardt method</th>
<th>PSO algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Iris plant</strong></td>
<td>0.001</td>
<td>0.005</td>
<td>0</td>
</tr>
<tr>
<td><strong>Wine</strong></td>
<td>1.24E-6</td>
<td>0.003</td>
<td>2817</td>
</tr>
<tr>
<td><strong>Object</strong></td>
<td>1.27E-6</td>
<td>2.84E-6</td>
<td>0</td>
</tr>
<tr>
<td><strong>Recognition</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>