01 Jan 2003

# Comparison of Particle Swarm Optimization and Backpropagation as Training Algorithms for Neural Networks

Ganesh K. Venayagamoorthy
*Missouri University of Science and Technology*

Venu Gopal Gudise



## Recommended Citation

G. K. Venayagamoorthy and V. G. Gudise, "Comparison of Particle Swarm Optimization and Backpropagation as Training Algorithms for Neural Networks," *Proceedings of the 2003 IEEE Swarm Intelligence Symposium, 2003. SIS '03*, Institute of Electrical and Electronics Engineers (IEEE), Jan 2003. The definitive version is available at https://doi.org/10.1109/SIS.2003.1202255

# Comparison of Particle Swarm Optimization and Backpropagation as Training Algorithms for Neural Networks

Venu G. Gudise and Ganesh K. Venayagamoorthy, *Senior Member, IEEE*
Department of Electrical and Computer Engineering
University of Missouri – Rolla, USA
*vgggwb@umr.edu and gkumar@ieee.org*

**Abstract - Particle swarm optimization (PSO) motivated by the social behavior of organisms, is a step up to existing evolutionary algorithms for optimization of continuous non-linear functions. Backpropagation (BP) is generally used for neural network training. Choosing a proper algorithm for training a neural network is very important. In this paper, a comparative study is made on the computational requirements of the PSO and BP as training algorithms for neural networks. Results are presented for a feedforward neural network learning a non-linear function and these results show that the feedforward neural network weights converge faster with the PSO than with the BP algorithm.**

## I. INTRODUCTION

The role of artificial neural networks in the present world applications is gradually increasing and faster algorithms are being developed for training neural networks. In general, backpropagation is a method used for training neural networks [1]-[5]. Gradient descent, conjugate gradient descent, resilient, BFGS quasi-Newton, one-step secant, Levenberg-Marquardt and Bayesian regularization are all different forms of the backpropagation training algorithm [6]-[10]. For all these algorithms storage and computational requirements are different, some of these are good for pattern recognition and others for function approximation but they have drawbacks in one way or other, like neural network size and their associated storage requirements. Certain training algorithms are suitable for some type of applications only, for example an algorithm which performs well for pattern recognition may not for classification problems and vice versa, in addition some cannot cater for high accuracy/performance. It is difficult to find a particular training algorithm that is the best for all applications under all conditions all the time.

A newly developed algorithm known as particle swarm is an addition to existing evolutionary techniques, which is based on simulation of the behavior of a flock of birds or school of fish. The main concept is to utilize the communication involved in such swarms or schools. Some of the previous work related to neural network training using the particle swarm optimization has been reported [11]-[15] but none have compared against conventional training techniques. In this paper, particle swarm optimization is compared with the conventional backpropagation (a gradient descent algorithm) for training a feedforward neural network to learn a non-linear function. The problem considered is how fats and how accurate can the neural network weights be determined by BP and PSO learning a common function. Detailed comparison of BP to PSO is presented with regard to their computational requirements.

The paper is organized as follows: In section II, the architecture of feedforward neural network considered in this paper is explained with the forward path and the backward path for the backpropagation method. In section III, a brief overview of the particle swarm optimization is given and its implementation is explained. Section IV describes how the optimal set of parameters for the PSO is determined. In section V, different neural network training procedures (incremental and batch) are described. In section VI, the results of training the neural network with BP and PSO are given and their performances are compared and contrasted.

## II. FEEDFORWARD NEURAL NETWORKS

Neural networks are known to be universal approximators for any non-linear function [16]-[17] and they are generally used for mapping error tolerant problems that have much data trained in noise. Training algorithms are critical when neural networks are applied to high speed applications with complex nonlinearities.

A neural network consists of many layers namely: an input layer, a number of hidden layers and an output layer. The input layer and the hidden layer are connected by synaptic links called weights and likewise the hidden layer and output layer also have connection weights. When more than one hidden layer exists, weights exist between such layers. Neural networks use some sort of "learning" rule by which the connections weights are determined in order to minimize the error between the neural network output and desired output. A three layer feedforward neural network is shown in Fig. 1.

### *A. Forward Path*

The feedforward path equations for the network in Fig. 1 with two input neurons, four hidden neurons and one output neuron are given below. The first input is $x$ and the second is a bias input ($1$). The activation function of all the hidden neurons is given by eq (1).

$$a_i = W_{ij} X \quad for\ i = 1\ to\ 4, j = 1, 2 \quad (1)$$

where $W_{ij}$ is the weight and $X = \begin{bmatrix} x \\ 1 \end{bmatrix}$ is an input vector.

The hidden layer output called the *decision* vector $d$ is calculated as follows for sigmoidal functions:

0-7803-7914-4/03/$10.00©2003 IEEE

$$d_i = \frac{1}{1 - e^{(a_i)}} \qquad for\ i = 1\ to\ 4 \quad (2)$$

The output of neural network $y$ is determined as follows:

$$y = [V_1, V_2, V_3, V_4] \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \end{bmatrix} \qquad (3)$$

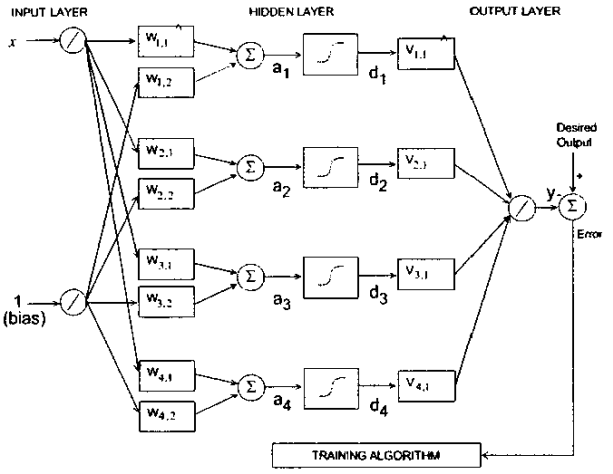


Fig. 1 Feedforward neural network with one hidden layer

### B. *Backward Path with Conventional Backpropagation*

The serious constraint imposed for the usage of backpropagation algorithm is that the hidden layer neuron function should be differentiable. If the inputs and desired outputs of a function are known then backpropagation can be used to determine weights of the neural network by minimizing the error over a number of iterations. The weight update equations of all the layers (input, hidden, output) in the multilayer perceptron neural network (MLPNN) are almost similar, except that they differ in the way the local error for each neuron is computed. The error for the output layer is the difference between the desired output (target) and actual output of the neural network. Similarly, the errors for the neurons in the hidden layer are the difference between their desired outputs and their actual outputs. In a MLP neural network, the desired outputs of the neurons in the hidden layer cannot be known and hence the error of the output layer is backpropagated and sensitivities of the neurons in the hidden layers are calculated.

The learning rate is an important factor in the BP. If it is too low, the network learns very slowly and if it is too high, then the weights and the objective function will diverge. So an optimum value should be chosen to ensure global convergence which tends to be difficult task to achieve. A variable learning rate will do better if there are many local and global optima for the objective function [18]. Backpropagation equations are explained in detail in [19] and they are briefly described below.

The output error $e_y$ is calculated as the difference between the desired output vector $y_d$ and actual output $y$.

$$e_y = y_d - y \qquad (4)$$

The decision error vector $e_d$ is calculated by backpropagating the output error $e_y$ through weight matrix $V$.

$$e_{di} = V_i^T e_y \quad for\ i = 1\ to\ 4 \quad (5)$$

The activation function errors are given by the product of decision error vector $e_{di}$ and the derivatives of the decision vector $d_i$ with respect to the activations $a_i$.

$$e_{ai} = d_i(1 - d_i)e_{di} \qquad (6)$$

The sensitivities (changes in the weights) are calculated as

$$\Delta V(k) = \gamma_m \Delta V(k-1) + \gamma_g e_y d^T \qquad (7a)$$

$$\Delta W(k) = \gamma_m \Delta W(k-1) + \gamma_g e_a X^T \qquad (7b)$$

where $\gamma_m$ is a momentum term, $\gamma_g$ is the learning gain and $k$ is the number of iteration. A momentum term produces a 1 filter effect in order to reduce abrupt gradient changes thus aiding learning. Finally the weight update equations are below.

$$W(k+1) = W(k) + \Delta W(k) \quad (8a)$$

$$V(k+1) = V(k) + \Delta V(k) \quad (8b)$$

## III. PARTICLE SWARM OPTIMIZATION

Particle swarm optimization is a form of evolutionary computation technique (a search method based on *natural* systems) developed by Kennedy and Eberhart [20]-[25]. PSO like a genetic algorithm (GA) is a population (swarm) based optimization tool. However, unlike in GA, crossover and mutation are carried out simultaneously in particle swarm. One major difference between particle swarm and traditional evolutionary computation methods is that particles' velocities are adjusted, while evolutionary individuals' positions are acted upon; it is as if the "fate" is altered rather than the "state" of the particle swarm individuals [25].

The system initially has a population of random solutions. Each potential solution, called particle, is given a random velocity and is flown through the problem space. The particles have memory and each particle keeps track of previous best position and corresponding fitness. The previous best value is called as '$p_{best}$'. Thus, $p_{best}$ is related only to a particular particle. It also has another value called '$g_{best}$', which is the best value of all the particles $p_{best}$ in the

swarm. The basic concept of PSO technique lies in accelerating each particle towards its $p_{best}$ and the $g_{best}$ locations at each time step. Acceleration has random weights for both $p_{best}$ and $g_{best}$ locations.

Figure 2 illustrates briefly the concept of PSO, where $P^k$ is current position, $P^{k+1}$ is modified position, $V_{ini}$ is initial velocity, $V_{mod}$ is modified velocity, $V_{pbest}$ is velocity considering $p_{best}$ and $V_{gbest}$ is velocity considering $g_{best}$.

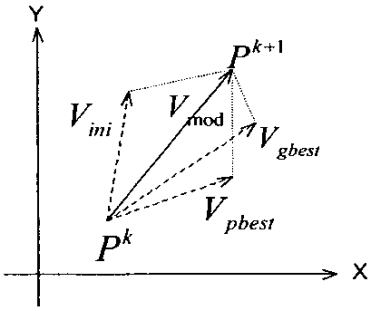


Fig. 2 Concept of changing a particle's position in PSO [26]

(i) Initialize a population (array) of particles with random positions and velocities of $d$ dimensions in the problem space.

(ii) For each particle, evaluate the desired optimization fitness function in $d$ variables.

(iii) Compare particle's fitness evaluation with particle's $p_{best}$. If current value is better than $p_{best}$, then set $p_{best}$ value equal to the current value and the $p_{best}$ location equal to the current location in $d$-dimensional space.

(iv) Compare fitness evaluation with the population's overall previous best. It the current value is better than $g_{best}$, then reset $g_{best}$ to the current particle's array index and value.

(v) Change the velocity and position of the particle according to equations (9) and (10) respectively. $V_{id}$ and $X_{id}$ represent the velocity and position of $i^{th}$ particle with $d$ dimensions respectively and, $rand_1$ and $rand_2$ are two uniform random functions.

$$V_{id} = W \times V_{id} + c_1 \times rand_1 \times (P_{bestid} - X_{id}) + c_2 \times rand_2 \times (G_{bestid} - X_{id}) \quad (9)$$

$$X_{id} = X_{id} + V_{id} \quad (10)$$

(vi) Repeat step (ii) until a criterion is met, usually a sufficiently good fitness or a maximum number of iterations/epochs.

PSO has many parameters and these are described as follows: $W$ called the inertia weight controls the exploration and exploitation of the search space because it dynamically adjusts velocity. Local minima are avoided by small local neighborhood, but faster convergence is obtained by larger global neighborhood and in general, global neighborhood is preferred. Synchronous updates are more costly than the asynchronous updates.

$V_{max}$ is the maximum allowable velocity for the particles. i.e. in case the velocity of the particle exceeds $V_{max}$ then it is reduced to $V_{max}$. Thus, resolution and fitness of search depends on $V_{max}$. If $V_{max}$ is too high, then particles will move beyond good solution and if $V_{max}$ is too low, then particles will be trapped in local minima. $c_1$, $c_2$ termed as cognition and social components respectively are the acceleration constants which changes the velocity of a particle towards $p_{best}$ and $g_{best}$ (generally somewhere between $p_{best}$ and $g_{best}$). Velocity determines the tension in the system. A swarm of particles can be used locally or globally in a search space. In the local version of the PSO, the $g_{best}$ is replaced by the $l_{best}$ and the entire procedure is same.

## IV. SELECTION OF PARAMETERS FOR PSO

The selection of these PSO parameters plays an important role in the optimization [24]. A single PSO parameter choice has a tremendous effect on the rate of convergence. For this paper, the optimal PSO parameters are determined by trial and error experimentations. Optimal here refers to the set of PSO parameters that yield the fastest neural network convergence.

The optimal PSO parameters determination is by varying the inertia weight $W$, whose dynamic range is between 0.2 and 1.2, maximum velocity $V_{max}$, search space range, social and cognitive coefficients and swarm size. Initial values for these parameters are taken to be: maximum velocity $V_{max}$ (4) and an initial search space range is selected between (-4, 4), number of particles in the swarm (20). The PSO algorithm with non-optimal parameters may diverge therefore, the PSO parameters and the number of iterations are only recorded when convergence is achieved at least 7 times in the 10 runs. It is found that as the inertia weight ($W$) is increased from 0.2, the number of times of convergence on average increased and the optimum inertia weight observed is between 0.7 and 0.8. This process is repeated for the $V_{max}$ varied from 0.2 to 10 with W = 0.8 and it is found that a $V_{max}$ of 2 gave the optimum results. Weights W of 0.7 and 0.8 performed equally well with velocity $V_{max}$ of 2, but the number of iterations required with W = 0.8 is lesser than that required with $W = 0.7$.

The search space range available for the particles plays an important role in converging to the solution. A small search space range of (-1, 1) did not provide enough freedom for the particles to explore the space and hence they failed to find the best position. As the search space range allowed is increased gradually from (-1, 1) to (-200, 200), it is found that a larger space helped the particles to achieve the global optimum much quicker. A search space range of (-100, 100) is observed to be the best range. With no limit on the search space range, the convergence rate decreased, with even cases

of no convergence. This is as a result of the particles exploring and not exploiting the optimum position.

Then the experiment is carried out with the optimal parameters obtained above for W, $V_{max}$ and search space range, now to determine the cognition ($c_1$) and social ($c_2$) coefficients. It is found that $c_1$=2 and $c_2$=2 gave best results (faster global convergence), also $c_1$=2.5 and $c_2$=1.3 gave good results.

With all these optimum values, the experiment is repeated varying the size of the swarm (no. of particles) from 1 to 1000. As the size increased from 20 to 25 and 25 to 30, there is an improvement in the convergence rate. The improvement for 20 to 25 is noticeable higher than from 25 to 30. When size is increased from 30 to 50, 50 to 100 and 100 to 1000, the performance is also improved at the cost of a higher computational time. A compromise between the computational time and the performance is a size of 25 particles for the swarm (for this example). The final optimal set of PSO parameters are:

| | |
|---|---|
| *Maximum velocity, $V_{max}$* | *2* |
| *Maximum search space range* | *(-100,100)* |
| *Inertia weight, W* | *0.8* |
| *Acceleration constants, $c_1$, $c_2$* | *2, 2* |
| *Size of swarm* | *25* |

## V. TRAINING PROCEDURES FOR NEURAL NETWORKS

There are two ways of presenting data to a neural network during training, namely: *batch* and *incremental* fashion and these are explained below.

### *A. Incremental Training*

In this method, each of the input pattern or training data is presented to the neural network and weights are updated for each data presented, thus the number of weight updates will be equal to the size of the training set. The inherent noise of this learning mode makes it possible to escape from undesired local minima of the error potential where the learning rule performs (stochastic) gradient descent. The noise is related to the fluctuations in the learning rule, which is a function of the weights.

### *B. Batch Training*

In this method, all of the input pattern or training data are presented to the neural network one after the other and then the weights are updated based on a cumulative error function. The process can be repeated over a number of iterations/epochs. In batch mode learning, the network gets struck in a local minimum, in where minimum only depends on the initial network state, the noise is homogeneous, i.e. same at each minimum. The neural network training comparison with BP and PSO carried out in this paper is based on the batch mode.

## VI. RESULTS

In order to compare the training capabilities of BP and PSO algorithms, a non-linear quadratic equation $y = 2x^2+1$, with data points (patterns) in range (-1, 1) presented in the batch mode to the feedforward neural network in Fig. 1. The flowchart procedure for implementing the PSO global version ($g_{best}$) is given in Fig. 3. The PSO parameters used in this study are those mentioned in Section IV above. For training a neural network using the PSO, the fitness value of each particle (member) of the swarm is the value of the error function evaluated at the current position of the particle and position vector of the particle corresponds to the weight matrix of the network. The vector e(particle) in Fig. 3 stores the minimum error encountered by the particles for the input patterns.

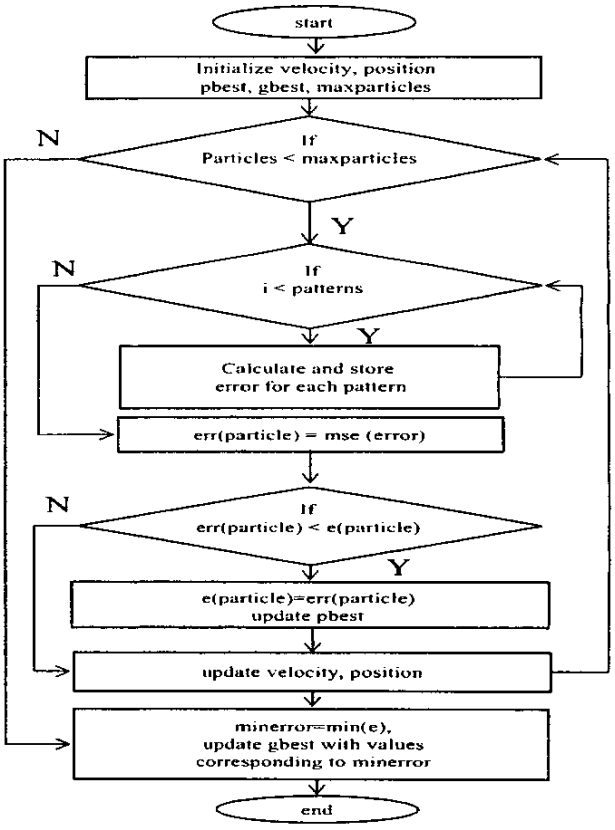


Fig. 3 Flowchart for training a feedforward NN using PSO

Table I shows the numerical values of total number of computations required for reaching an error goal of 0.001. The value of bias is taken in this case to be 1. The values shown were determined after averaging the values of 10 different trainings. Fig. 4 shows the mean square error (MSE) versus the number of iterations with BP and PSO

during training with bias value 1. Fig. 5 shows an expanded view of Fig. 4. It is clear that with the PSO the MSE drops to under 0.01 in few iterations unlike with BP. Fig. 6 shows the test plots after the neural network is trained with BP and PSO and subjected to the input vectors between -1 and 1. Fig. 7 shows an expanded view of Fig. 6 and it is clear that the neural network trained with the PSO algorithm approximates the nonlinear function better than the one trained with BP. This means that PSO yields a better global convergence than a conventional BP. Fig 8 shows the mean square error (MSE) versus the number of iterations with BP and PSO during training for the feedforward neural network with bias of 2. The bias has helped the BP algorithm performance but the PSO results are now obtained with 8 particles. It is observed from Table II that still PSO out performs the BP despite a change in bias.

For training a feedforward neural network similar to Fig. 1 of size $n \times m \times r$, where $n$ is the number of the inputs, $m$ is the number of hidden layer neurons and $r$ is the number of the outputs, Tables III and IV show the number of computations (multiplications and additions) required with the BP and PSO training algorithms respectively for the batch mode learning. These tables give an idea on expectations of the computational demands.

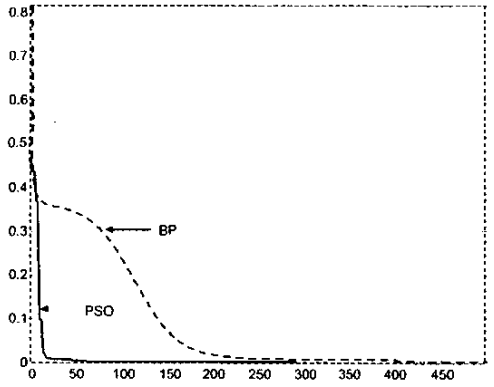


Fig. 4 Mean square error curves of neural networks during training with BP and PSO for bias 1

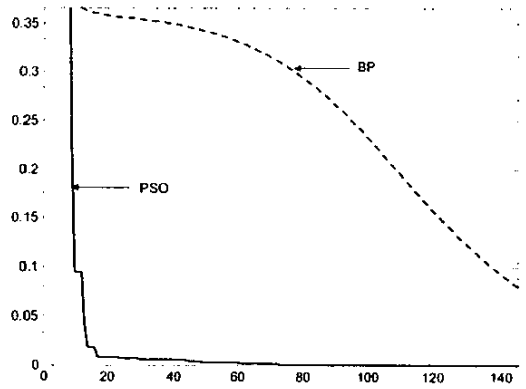


Fig. 5 Expanded view of Fig. 4

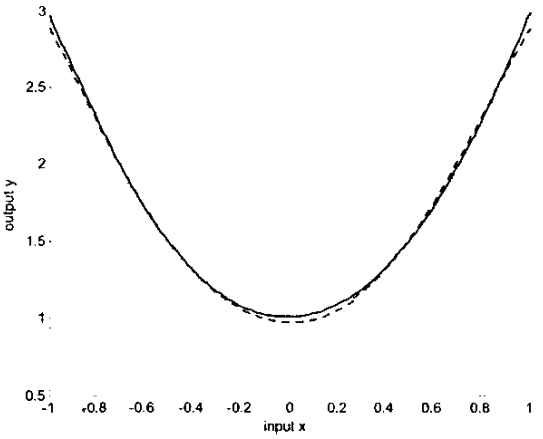


Fig. 6 Test curves for trained neural networks with fixed weights obtained from BP and PSO training algorithms with bias 1

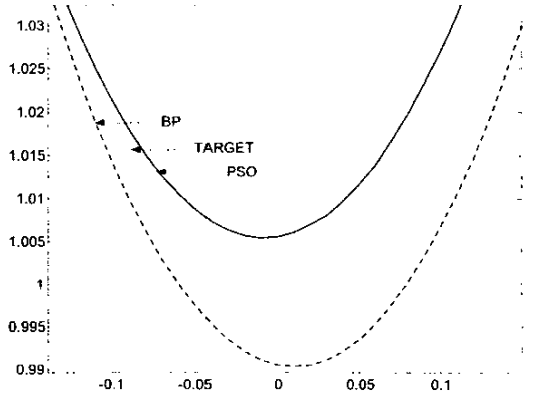


Fig. 7 Magnified view of the above test curves

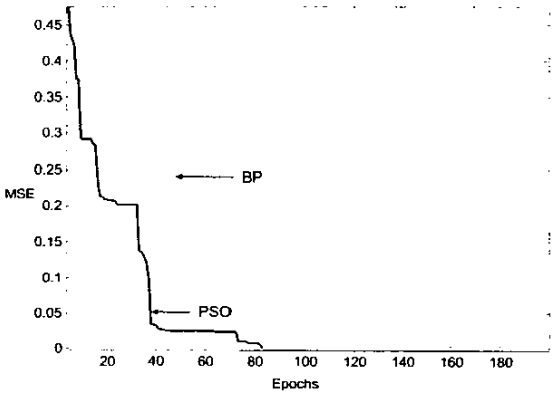


Fig. 8 Mean square error curves of neural networks during training with BP and PSO for bias 2

TABLE I
COMPARISION OF NUMBER OF COMPUTATIONS (MULTIPLICATIONS AND ADDITIONS) INVOLVED IN TRAINING A FEEDFORWARD NEURAL NETWORK OF SIZE $2 \times 4 \times 1$ FOR BP AND PSO (WITH BIAS 1 )

| Error=0.001 | PSO | BP | PSO | BP |
|---|---|---|---|---|
| Patterns (input $x$) | -1:0.1:1 | | -1:0.01:1 | |
| Iterations | 194 | 9836 | 116 | 962 |
| Forward path (additions+ multiplications) | 2546250 | 4750788 | 14572500 | 4447326 |
| Backward path (additions+ multiplications) | 523800 | 14045808 | 835200 | 13148616 |
| Total (Forward + Backward) | 3070050 | 18796596 | 15407700 | 17595942 |
| Ratio of computations | 6.1226 | | 1.1420 | |

TABLE II
COMPARISION OF NUMBER OF COMPUTATIONS (MULTIPLICATIONS AND ADDITIONS) INVOLVED IN TRAINING A FEEDFORWARD NEURAL NETWORK OF SIZE $2 \times 4 \times 1$ FOR BP AND PSO (WITH BIAS 2)

| Error=0.001 | PSO | BP |
|---|---|---|
| Patterns (input $x$) | -1:0.1:1 | |
| Iterations | 194(83 | 9836(307 |
| Forward path (additions+ multiplications) | 348600 | 148281 |
| Backward path (additions+ multiplications) | 71712 | 438396 |
| Total (Forward + Backward) | 420312 | 586677 |
| Ratio of computations | 1.3958 | |

TABLE III
NUMBER OF COMPUTATIONS (MULTIPLICATIONS AND ADDITIONS) INVOLVED IN TRAINING A FEEDFORWARD NEURAL NETWORK OF SIZE $n \times m \times r$ WITH BP

| Equation No. | Multiplications | Additions | Sigmoids |
|---|---|---|---|
| 1 (activation) | patterns*(m*n) | patterns*m*(n-1) | 0 |
| 2 (sigmoid) | 0 | 0 | patterns*m |
| 3 (output) | patterns*(r*m) | patterns*r*(m-1) | 0 |
| Forward path | patterns*(m*(r+n)) | patterns*(m*(r+n)-(m+r)) | patterns*m |
| 4 (output error) | 0 | patterns*n | 0 |
| 5 (decision error) | patterns*(r*m) | patterns*r*(m-1) | 0 |
| 6 (activation error) | patterns*(2*m) | patterns*m | 0 |
| 7a ($\Delta$v) | patterns*(r*(2*m+1)) | patterns*(r*m+r*(m-1)) | 0 |
| 7b ($\Delta$w) | patterns*(m*(2*n+1)) | patterns*(m*n+m*(n-1)) | 0 |
| 8 | 0 | patterns*2 | 0 |
| Training | patterns*(m*(3*r+2*n+2)+m+r) | patterns*(m*(3*r+2*n+1)-(m+r)) | 0 |
| Total computations | patterns*(m*(r+n))+patterns*(m*(3*r+2*n+2)+m+r) | patterns*(m*(r+n)-(m+r))+patterns*(m*(3*r+2*n+1)-(m+r)) | patterns*m |

TABLE IV
NUMBER OF COMPUTATIONS (MULTIPLICATIONS AND ADDITIONS) INVOLVED IN TRAINING A FEEDFORWARD NEURAL NETWORK OF SIZE $n \times m \times r$ WITH PSO

| Equation No | Multiplications | Additions | Sigmoids |
|---|---|---|---|
| 1 (activation) | particles*patterns*m*n | Particles*patterns*(m*(n-1)) | 0 |
| 2 (sigmoid) | 0 | 0 | particles*patterns*m |
| 3 (output) | particles*patterns*r*m | Particles*patterns*(r*(m-1)) | 0 |
| Forward path | particles*patterns*(m*(r+n)) | Particles*patterns*(m*(r+n)-(m-r)) | particles*patterns*m |
| 4 (output error) | 0 | Particles*patterns | 0 |
| 5 (min error) | 0 | Particles | 0 |
| 9 (velocity) | particles*(3*m*(n+r)+2) | Particles*3*m*(n+r) | 0 |
| 10 (position) | 0 | Particles*m*(n+r) | 0 |
| Training | particles*(3*m*(n+r)+2) | Particles*(patterns+m*(4*n+4*r)+1) | 0 |
| Total computations | particles*patterns*(m*(r+n))+ particles*(3*m*(n+r)+2) | particles*patterns*(m*(r+n)-(m-r))+particles*(patterns+m*(4*n+4*r)+1) | particles*patterns*m |

## VII. CONCLUSIONS

A feedforward neural network learning a nonlinear function with the backpropagation and particle swarm optimization algorithms have been presented in this paper. The number of computations required by each algorithm has shown that PSO requires less to achieve the same error goal as with the BP. Thus, PSO is a better one for applications that require fast learning algorithms. An important observation made is that when the training points are fewer, the neural network learns the nonlinear function with six times lesser number of computations with PSO than that required by the BP and in other cases, comparable performance is observed. Moreover, the success of BP depends on choosing a bias value unlike with PSO. The use of PSO as a training algorithm for recurrent neural networks has been studied and similar results are also found. Further work is to investigate using PSO to optimize PSO parameters for neural network training and other tasks. The concept of the PSO can be incorporated into BP algorithm to improve its global convergence rate. This is currently being studied for online learning which is critical for adaptive real time identification and control functions.

## REFERENCES


[1] P.J. Werbos, "Backpropagation through time: what it does and how to do it". *Proceedings of the IEEE*, Vol: 78:10, Page(s): 1550 -1560, Oct. 1990

[2] P.J. Werbos, The Roots of Backpropagation, New York: Wiley, 1994

[3] D.E.Rumelhart, G.E. Hinton, and R.J. Williams, "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing*, vol.1, cahp.8, eds. D.E.Rumelhart and J.L. McClelland, Cambridge, MA:M.I.T Press, pp.318-62, 1986

[4] D.E.Rumelhart, G.E. Hinton and R.J. Williams, "Learning representations by Back-propagating errors", *Nature*, Vol.323, pp.533-6, 1986

[5] M.T. Hagan, H.B. Demuth and M. Beale, Neural Network Design, Boston, MA: PWS Publishing Company, 1996

[6] R. Battit, "First and second order methods of learning: between the steepest descent and Newton's method", *Neural Computation*, Vol.4:2, pp.141-66, 1991

[7] C.H. Chen and H. Lai, "An Empirical study of the Gradient descent and the Conjugate gradient backpropagation neural networks" . Proceedings OCEANS '92. Mastering the Oceans Through Technology. Vol: 1 pp. 132 -135, 1992

[8] M.F. Moller, A Scaled conjugate gradient algorithm for fast supervised learning PB-339 Reprint, Computer Science Department, University of Aaurhus, Denmark, 1990

[9] R. Rosario, A Conjugate-gradient based algorithm for training of the feed-forward neural networks, Ph.D. Dissertation, Melbourne: Florida Institute of Technology, September 1989

[10] M.T. Hagan and M.B. Menhaj, "Training feedforward networks with the Marquardt algorithm" *IEEE Transactions on Neural Networks*, Vol: 5 : 6, pp. 989 -993, Nov. 1994

[11] J. Salerno, "Using the particle swarm optimization technique to train a recurrent neural model", IEEE International Conference on Tools with Artificial Intelligence, pp.45-49, 1997

[12] C. Zhang, Y. Li, and H. Shao, "A new evolved artificial neural network and its application", Proceedings of the 3rd World Congress on Intelligent Control and Automation, Hefei, China, Vol. 2, pp.1065-1068, June 2000

[13] C. Zhang, H. Shao, and Y. Li, "Particle swarm optimisation for evolving artificial neural network", Proceedings of the IEEE

International Conference on Systems, Man, and Cybernetics . Vol. 4, pp.2487-2490, 2000

[14] M. Settles and B. Rylander, "Neural network learning using particle swarm optimizers", Advances in Information Science and Soft Computing, pp.224-226, 2002

[15] F. Van den Bergh. and A.P. Engelbrecht., "Cooperative learning in neural networks using particle swarm optimizers," *South African Computer Journal*, Vol. 26 pp. 84-90, 2000

[16] S. Lawrence, A. C. Tsoi and A. D. Back, "Function approx- imation with neural networks and local methods: bias, variance and smoothness", Proceedings of the Australian Conference on Neural Networks ACNN 96,Canberra, Australia, pp. 16-21, 1996

[17] Meng Jinli and Sun Zhiyi, "Application of combined neural networks in nonlinear function approximation." Proceedings of the 3rd World Congress on Intelligent Control and Automation, Vol.2, pp. 839 -841, 2000

[18] M.H. Ham and I. Kostanic, Principles of Neurocomputing for Science and Engineering, McGraw-Hill, INC. NY, ISBN: 0070259666, 2001

[19] B. Burton and R. G. Harley, "Reducing the computational demands of continually online trained artificial neural networks for system identification and control of fast processes", in Conf. Rec. IEEE IAS Annual Meeting, Denver, CO, pp. 1836–1843, Oct. 1994

[20] James Kennedy and R. Eberhart, "Particle swarm optimization". Proceedings, IEEE International Conf. on Neural Networks, Perth, Australia. Vol. IV, pp. 1942–1948

[21] Y. Shi and R. Eberhart, "Empirical study of particle swarm optimization". Proceedings of the 1999 Congress on Evolutionary Computation, CEC 99., Vol. 3

[22] Y Shi and R. Eberhart, "A modified particle swarm optimizer". IEEE International Conf. on Evolutionary Computation, Anchorage, Alaska, USA, pp. 69 –73, May 1998

[23] R. Eberhart and Y. Shi, "Particle swarm optimization: developments, applications and resources". Proceedings of the 2001 Congress on Evolutionary Computation, Vol: 1, pp. 81 -86, 2001

[24] Y. Shi and R. Eberhart, "Parameter Selection in Particle Swarm Optimization". Proc. Seventh Annual Conf. on Evolutionary Programming, pp.591-601, March 1998

[25] James Kennedy, Russell C. Eberhart, Yuhui Shi, Swarm intelligence, Morgan Kaufmann Publishers, 2001

[26] H. Yoshida, Y. Fukuyama, S. Takayama. and Y. Nakanishi., "A particle swarm optimization for reactive power and voltage control in electric power systems considering voltage security assessment." IEEE SMC '99 Conf. Proceedings. Vol: 6, pp. 497 -502, 1999