Simulation Design of a Backpropagation Neural System of Sensor Network Trained by Particle Swarm Optimization

Prof. Dr. Hanan A. R. Akkar, Assist. Prof. Dr. Aied K. AL-Samarrie, Azzad B. Saeed

Abstract— The sensor network consists of two main units, they are: the sensor unit and the artificial intelligent system. The sensor unit is used for converting the output analog signal of the sensor to binary data, while the artificial intelligent system is used for processing these binary data and presenting a final decision. In this paper, a Backpropagation Neural Network is designed and simulated, which is a powerful artificial intelligent system, and it can be trained by an optimization method for updating the weights and biases of the hidden and output layers. *Satlins* and *Satlin* functions had been used as linear activation functions for the hidden and output layers. *Trainpso* function had been used as a training function for the proposed system, which is a particle swarm optimization method of training. It is worth to mention, that no previous research used these three functions together for such analysis. This system had been simulated and tested using MATLAB software package, the testing process had offered stimulant results, and the actual output data had fitted the desired output data, while the Mean Square Error had reached to zero with 56 iterations for getting best value of solution, where, no previous research had reached to this optimal result for such design.

Index Terms— Artificial Intelligent (AI), Backpropagation, Mean Square Error (MSE), Particle Swarm Optimization (PSO), MATLAB, Sensor Network(SN), Trainpso.

IJSER

1 INTRODUCTION

THE *Backpropagation Neural Network* is one of the effective artificial intelligent systems used in the sensor networks today[1].

It is a common method of training artificial neural networks used in conjunction with an optimization method such as gradient descent. The method calculates the gradient of a loss function with respect to all the weights in the network. The gradient is fed to the optimization method which in turn uses it to update the weights, in an attempt to minimize the loss function[2].

Backpropagation requires a known, desired output for each input value in order to calculate the loss function gradient. It is therefore, usually considered to be a method. It is a generalization of the delta rule to multi-layered feed forward networks, made possible by using the chain rule to iteratively compute gradients for each layer. Backpropagation requires that the activation function used by the artificial neurons or (nodes) be differentiable[3].

- *Hanan A. R. Akkar is currently Professor has Ph.D degree in electronic engineering in University of Technology, Iraq.*
- *Aied K. AL-Samarrie is currently Assist. Professor has Ph.D degree in communication engineering in University of Technology, Iraq.*
- *Azzad B. Saeed is currently Lecturer has Msc. degree in electronic engineering in University of Technology, Iraq.*

By applying of the backpropagation algorithm, every iteration of training follows these steps: 1) a specific case of training data must be entered through the network in a forward direction, presenting results at the output layer, 2) the error values must be calculated at the output nodes based on known desired output data, and the essential changes to the input connection weights of the output layer can be determined based on this error values, 3) the variation of the input connection weights of the preceding network layers can be determined as a function of the properties of the neurons to which they directly connect (connection weight changes can be calculated, layer by layer, as a function of the errors achieved for all subsequent layers, applying backward to the direction of the input layer) until all essential weight changes are calculated completely for the overall network[4].

The calculated weight changes are then performed throughout the entire network, the new iteration starts, and the overall procedure is repeated again using the next training pattern data. For a neural network with hidden layers, the backpropagation algorithm can be given by the three Equations (1),(2),(3), where i is the emitter layer of nodes, j is the receiver layer of nodes, k is the layer of nodes that follows j layer, ij is the layer of weights between node layers i and j, jk is the layer of weights between node layers j and k, weights are represented by(W), node activation functions are represented by (a), delta values for nodes are represented by (δ), and (ε) is the learning rate[5]:

$$\Delta W_{ijm} = \varepsilon \delta_{jp} a_{iq} \quad (1)$$

$$\delta_{jp} = a_{jp}(1 - a_{jp})(t_{jp} - a_{jp}) \quad (2)$$

This equation used for the output nodes.

$$\delta_{jp} = a_{jp}\left(1 - a_{jp}\right)\sum_{x=0}^{n}\delta_{kx}w_{jkx} \quad (3)$$

This equation used for the hidden nodes.

Equation (1) says that the change in a specific weight (m) stayed between layers i and j is equated to the production of: 1) the learning rate (ε); 2) the delta value for node p in layer j; and 3) the activation function (a) of node q in layer i. Practically, the learning rate (ε) is typically proposed a range $0 \leq \varepsilon \leq 0.1$; higher values may produce faster convergence on a solution, and may also increase the instability and may lead to a failure to converge. The delta value(δ_{jp}) for node p in layer j in Equation (1) can be given either by Equation (2) or by Equation (3), according to whether or not the node is in an output or hidden layer[5].

Equation (2) determines the delta value (δ_{jp}) for node p of layer j if node p is an output node, with the understanding that a sigma activation function is used here as a non-linear activation function. Equation (3) determines the delta value (δ_{jp}) for node p of layer j if node p is a hidden node. This equation states that the delta value of a specific node of interest is a function of: 1) the activation function at the same node, 2) the sum of the products of the delta values of relevant nodes in the succedent layer with the connections weights associated with the vectors that connect the nodes[5].

2 Particle Swarm Optimization (PSO)

In 1995, Dr. Eberhart and Dr. Kennedy had developed Particle Swarm Optimization (PSO), which is a population based stochastic optimization technique inspired by social behavior of fish schooling or bird flocking[6].

PSO had shared many analogies with evolutionary computation techniques such as Genetic Algorithms (GA). The system is started with a population of random solutions and searches for optima by generations updating. However, unlike GA, PSO does not have evolution operators such as crossover and mutation. In PSO, the particles (which is the potential solutions) fly through the problem zone by following the current optimum particles.

The advantages of PSO compared to GA, are that:1) PSO is easy for implementing and there are few parameters must be adjusted.2) PSO had been successfully applied in many areas, such as in function optimization, artificial neural network training, fuzzy system controlling, and other areas where GA can be applied[7].

PSO is learned from the scenario, which had used it to solve the optimization problems. In PSO, a (bird) represents each single solution in the search zone, which called (particle). All the particles have fitness values which had been evaluated by the fitness function to be optimized, and they have velocities used to direct the particles flying. The particles fly through the problem zone by following the current optimum particles[7].

PSO is began with a group of random particles (solutions) and then searches for optima by generations updating. For each iteration, every particle is updated by following dual (*best*) values. The first is the best solution (fitness) it has achieved so far. (The fitness value is also stored.) This value is called *pbest*. The second (*best*) value that is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the population. This best value is a global best and called *gbest*. When a particle takes part of the population as its topological neighbors, the best value is a local best and is called *lbest*[8].

After finding the two best values, the particle updates its velocity and positions with following Equations (4) and (5)[9].

$$Velocityi(t) = w.velocityi(t-1) + c1.rand1\big(pbesti(t)\big) - positioni(t-1) + c2.rand2\big(gbesti - positioni(t-1)\big) \quad (4)$$

$$Positioni(t) = positioni(t-1) + velocityi(t) \quad (5)$$

Where w represents the inertia weight, $c1$ and $c2$ represent acceleration constants, where, $c1 = c2 = 2$, and r represents a random function in the range [0, 1]. For equation (4), the first part corresponds the inertia of pervious velocity; the second part is the "cognition" part, which corresponds the private thinking by itself; the third part is the "social" part, which corresponds the cooperation among the particles. *pbesti* is the personal best position has been recorded by particle i, while *gbesti* is the global best position has been obtained by any particle in the population.

PSO process is briefly explained in the following steps:
1)For each particle, Initialize particle.
2)For each particle, Calculate fitness value, then, If the fitness value is better than the best fitness value (*pbest*) set current value as the new *pbest*.
3) Choose the particle with the best fitness value of all the particles as the *gbest*, then for each particle: a) Calculate particle velocity according Equation (4). b) Update particle position according Equation (5).

While maximum iterations or minimum error criteria or maximum iterations isn't achieved Particles' velocities for every dimension are clamped to a maximum velocity *Vmax*. If the sum of accelerations causes the velocity of that dimension to exceed *Vmax* (which is a parameter specified by the user), Then the velocity of that dimension is limited to *Vmax*[9].

3 Related Works

There are several previous works had applied the Particle Swarm Optimization (PSO) algorithm in the sensor network applications, where, they had used this algorithm for solving network routing problems or addressing Wireless Sensor Network issues such as optimal deployment, node localization, clustering and data-aggregation. But the proposed work had differed from all previous works by using the PSO algorithm as a training method for updating the weights connections and biases of

the proposed neural network (the intelligent system) of the sensor network for reaching to zero mean square error, which is optimal result.

Te-Jen, Ming-Yuan, and Yuei-Jyne[10] had proposed a control of the coverage problem optimization via the adaptive particle swarm optimization (APSO) approach. The proper selection of inertia weight of APSO gives balance between global and local searching, and they had showed that the larger weight helps to increase convergence speed while the smaller one benefits convergence accuracy, decreasing the algorithm operation times.

K. kavitha and M. Mohamed[11] had used particle swarm optimization (PSO), a metaheuristic algorithm to perform the process of routing. Since PSO does not have a defined fitness function, they incorporate user defined QoS parameters to define the fitness function.

Haiping, Junqing, Ruchuan, and yishing[12] had proposed algorithm combines the ionic bond method with particle swarm optimization (PSO), where ionic bond method uses a judicious ionic bond between two sensor nodes of sensor network to determine which node needs to move and also the path and direction of the movement and PSO suitable for solving multi-dimension function optimization in continuous space.

4 Simulation Design of the Proposed System

The proposed system used in this design, was a back propagation neural network, its design and simulation had been realized using MATLAB package with the following considerations, the sensor network consists of two sensor units, each sensor unit has two bits of binary output data, which driven to the input lines of the proposed system, so the proposed system must have four input lines. The output of the sensor unit has three binary states, the first is the binary data (01) represents the (LOW) state, the second binary data is (10) represents the (MEDIUM) state, and the third binary data is (11) represents the (HIGH) state.

The output of the proposed system has three logic lines, one can choose three possible states for the desired output data of this system, the first desired output data is (001), which represents the (LOW) state, the second desired output data is (010), which represents the (MEDIUM) state, and the third desired output data is (100), which represents the (HIGH) state. One can conclude from these considerations that the proposed simulation system has four logic input lines and three logic output lines.

The desired output data of the proposed system, must be equated to the average of the input data introduced by the sensor units according to the Table (1), for example, if the input data introduced by sensor unit (1) is (01) (LOW) state, and the input data introduced by sensor unit (2) is (01) (LOW) state too, then the proposed system will produce an output data (001) (LOW) state as shown in Table (1), and so on.

TABLE 1
THE RELATION BETWEEN INPUT AND DESIRED OUTPUT DATA

Input Data				Desired Output Data		
Sensor (1)		Sensor (2)				
a	b	c	d	x	y	z
1	0	1	0	1	0	0
0	1	1	0	0	1	0
1	1	1	0	0	1	0
1	0	0	1	0	1	0
0	1	0	1	0	1	0
1	1	0	1	0	0	1
1	0	1	1	0	1	0
0	1	1	1	0	0	1
1	1	1	1	0	0	1

The input layer of the proposed system must have four neurons, because it has four input lines, and the output layer must have three neurons, because it has three output lines, and the hidden layer must have ten neurons for obtaining accurate results. Linear activation functions had been chosen for the hidden and output layers as shown in Figure (1) , *Satlins* activation function for the hidden layer, and *Satlin* activation function for the output layer. The MATLAB function (*Trainpso*) had been chosen as a learning function for the proposed system, it means particle swarm optimization learning function, used for updating the weights and biases of the hidden and output layers, which is a very fast training method.

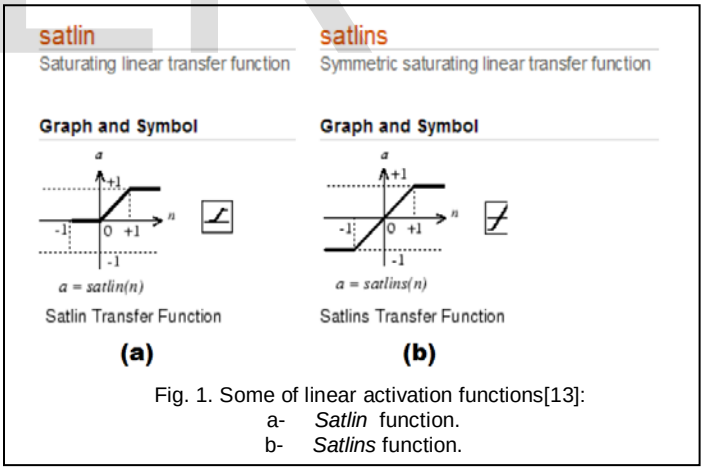

Fig. 1. Some of linear activation functions[13]:
a- *Satlin* function.
b- *Satlins* function.

Particle swarm optimization had been used a learning method for updating the weights and biases of the back propagation neural network of the proposed system, which it had designed and simulated with the following considerations:

1. Each particle has {(4×10)+(10×3)}=70 values of total network weights.
2. Population size = 25 particles.
3. Maximum no. iteration = 2000.
4. Minimum error gradient = 1×10^{-9}.
5. Initial total weights in the range = (0.4,0.9).

6. Searching range for input weights of hidden layer = (-100,100).
7. Searching rang for input weights of output layer = (-100,100).
8. Searching range for bias = (-8,8).

Finally, The previous considerations had helped the programmer to write an appropriate MATLAB software using back propagation neural instructions. This software had been finished by the instruction {gensim(net)} to generate the proposed system block. After generation of this block, the input ports and a multiplexer had been connected to the input line of the system, while, the output ports and a demultiplexer had been connected to the output line of the same system. The using of input and output ports had made the connection of the proposed system to external devices easier. The flowchart of training the proposed system had realized as shown in Figure (2). From this figure, one can see that input training data must be entered firstly, and then the desired output data. The activation and training functions must be entered, then the learning rate and maximum no. epochs must be set. Finally, the training must be applied to the proposed system for getting the actual output.

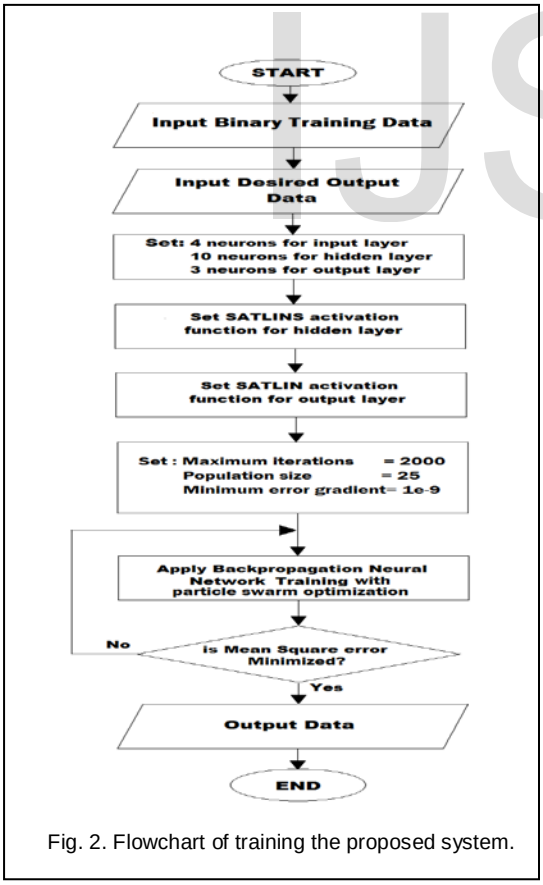

Fig. 2. Flowchart of training the proposed system.

5 Simulation Design of the Proposed System

The proposed system used in this design, was a back propagation neural network, its design and simulation had been realized using MATLAB package with the following considerations, the sensor network consists of two sensor units, each sensor unit has two bits of binary output data, which driven to the input lines of the proposed system, so the proposed system must have four input lines. The output of the sensor unit has three binary states, the first is the binary data (01) represents the (LOW) state, the second binary data is (10) represents the (MEDIUM) state, and the third binary data is (11) represents the (HIGH) state.

The output of the proposed system has three logic lines, one can choose three possible states for the desired output data of this system, the first desired output data is (001), which represents the (LOW) state, the second desired output data is (010), which represents the (MEDIUM) state, and the third desired output data is (100), which represents the (HIGH) state. One can conclude from these considerations that the proposed simulation system has four logic input lines and three logic output lines.

The desired output data of the proposed system, must be equated to the average of the input data introduced by the sensor units according to the Table (1), for example, if the input data introduced by sensor unit (1) is (01) (LOW) state, and the input data introduced by sensor unit (2) is (01) (LOW) state too, then the proposed system will produce an output data (001) (LOW) state as shown in Table (1), and so on.

The input layer of the proposed system must have four neurons, because it has four input lines, and the output layer must have three neurons, because it has three output lines, and the hidden layer must have ten neurons for obtaining accurate results. Linear activation functions had been chosen for the hidden and output layers as shown in Figure (1) , *Satlins* activation function for the hidden layer, and *Satlin* activation function for the output layer. The MATLAB function (*Trainpso*) had been chosen as a learning function for the proposed system, it means particle swarm optimization learning function, used for updating the weights and biases of the hidden and output layers, which is a very fast training method.

Particle swarm optimization had been used a learning method for updating the weights and biases of the back propagation neural network of the proposed system, which it had designed and simulated with the following considerations:

1. Each particle has {(4×10)+(10×3)}=70 values of total network weights.
2. Population size = 25 particles.
3. Maximum no. iteration = 2000.
4. Minimum error gradient = 1×10^{-9}.
5. Initial total weights in the range = (0.4,0.9).
6. Searching range for input weights of hidden layer = (-100,100).
7. Searching rang for input weights of output layer = (-100,100).
8. Searching range for bias = (-8,8).

Finally, The previous considerations had helped the programmer to write an appropriate MATLAB software using back propagation neural instructions. This software

had been finished by the instruction {gensim(net)} to generate the proposed system block. After generation of this block, the input ports and a multiplexer had been connected to the input line of the system, while, the output ports and a demultiplexer had been connected to the output line of the same system. The using of input and output ports had made the connection of the proposed system to external devices easier. The flowchart of training the proposed system had realized as shown in Figure (2). From this figure, one can see that input training data must be entered firstly, and then the desired output data. The activation and training functions must be entered, then the learning rate and maximum no. epochs must be set. Finally, the training must be applied to the proposed system for getting the actual output.

6 Results and Discussion

After executing the proposed MATLAB software, a simulation system block will be generated, this block has four input and three output lines, the input lines must be connected to the input ports, and the output lines must be connected to output ports, the data type of the input and output ports must be converted to the Boolean type. A multiplexer must be connected between the input ports and the input of the simulation system block, and a demultiplexer must be connected between the output of the simulation system block and the output ports.

After opening the system block, a system network will be exhibited as shown in Figure (3), this network constructed from two layer blocks, these are : layer (1) and layer (2) blocks. Layer (1) block represents the input layer and the hidden layer, while layer (2) block represents the output layer.

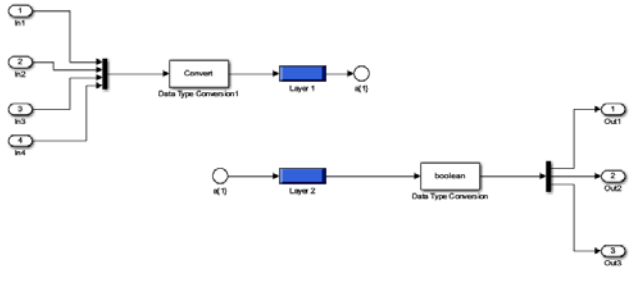

Fig. 3. Internal network of the proposed system block.

After opening the layer (1) block, a system network will be exhibited, which constructed from: delays(1),weight, bias, summation, and activation function blocks, the activation function of this layer is *Satlins* function. After opening the weight block of the this layer, a system network will be exhibited as shown in Figure (4), this network represents the hidden layer, which constructed from ten neurons with their updated weights.

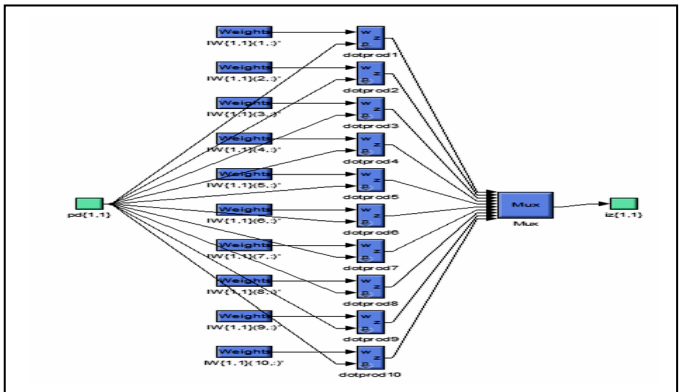

Fig. 4. Internal network of weight block of layer(1) block.

After opening the layer (2) block, a system block will be exhibited, which constructed from: delays(1), weight, bias , summation, and activation function blocks, the activation function of this layer is *Satlin* function, and after opening the weight block of this layer, a system network will be exhibited as shown in Figure (5), which constructed from three neurons of the output layer with their updated weights.

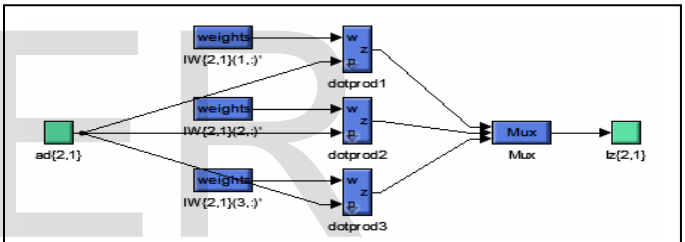

Fig. 5. Internal network of Weight block of layer(2) block.

The data type of the overall simulation block must be converted from (double) data type to fixed data (fixdt) type, and the data type converter blocks must be connected at the inputs and outputs of the this system, whereas, a data type converter must be connected between the input ports and the input of the simulation system block, which converts the Boolean data type of the input ports to the fixed data (fixdt) type of the simulation system block, while, another data type converter must be connected between the output of the simulation system block and the output ports, which converts the fixed data (fixdt) type of the simulation system block to the Boolean data type of the output ports.

The software program had been executed to present two result windows as shown in Figures (6) & (7), the first one shows two graphs, the left graph represents the connection diagram of the proposed back propagation neural network, there are four neurons at the input layer, ten neurons at the hidden layer, three neurons at the output layer, and 70 connections among them. The right graph shows the relation between the mean square errors versus no. iterations, as shown, the relation line non-linearly failed to (zero) value with 48 iteration, i.e. the complete learning

had been achieved to get best value of solution with zero mean square error at iteration (49). The input weights of the hidden layer was at range (-97.6639, 100), and the input weights of the output layer was at range (-100,100), while the biases was at range (-6.9776, 4.5696).

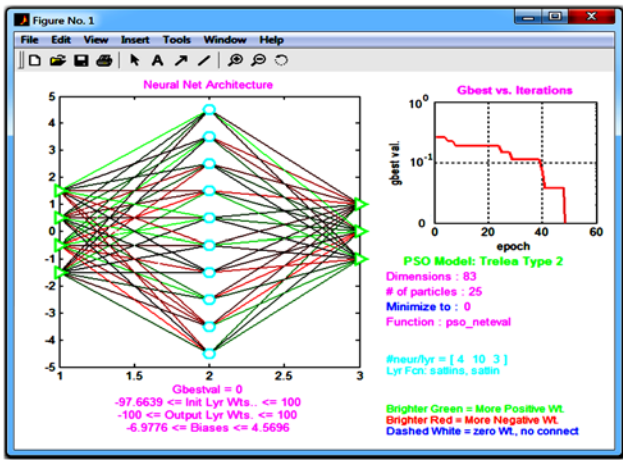

Fig. 6. The result window of executing the proposed software program.

Figure (7) shows the result of another training of the proposed network (or another executing the software program of the proposed system). There are two graphs in

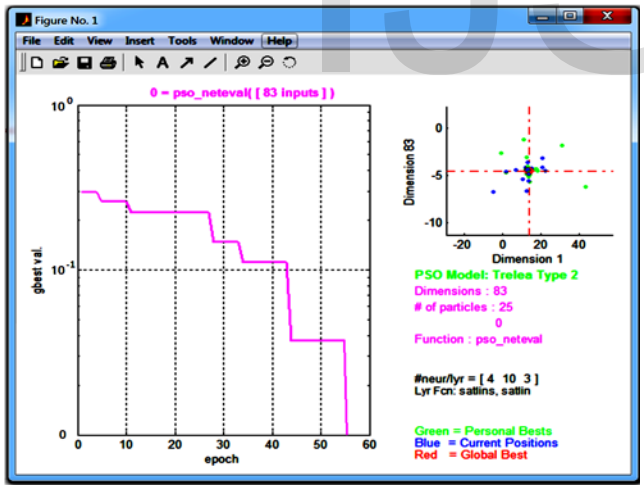

Fig. 7. Another result window shows the accumulation of the particles.

this figure, the left one represents the relation between Mean Square Error versus no. iteration, as shown, the relation line of this graph had reached to (zero) value at iteration 56 for getting the best value of solution. The right graph of this figure represents the tree type (2) PSO model, which shows the accumulation of the particles around the intersection of the red sporadic lines. This center represents the best solution of the problem, whereas, these particles stayed in the center of these lines at the end of the best learning process.

7 Conclusions

For increasing the sensor units of the proposed system, the input lines of the system will increase too, which leads to increasing the neurons of the input layer, and this process makes the proposed system more complex, and increases the size of the simulation software.

Practically, only the linear activation functions must be used for the hidden and output layers of the proposed network, whereas, the non-linear activation functions cannot be used with the proposed system, because the MATLAB package cannot convert this system to VHDL code program, and then it cannot be downloaded into an FPGA .

One can proposes more output levels at the output layer for increasing the accuracy of the output results, but this process leads to increase the neurons of the output layer, which makes the simulation system more complex, and enlarges the size of the system software.

Increasing the hidden layers, or decreasing the neurons of the single hidden layer of the proposed system, leads to decreasing the accuracy of the results.

References

[1] Subhas C. M., Henry L.," ADVANCED IN WIRELESS SENSORS AND SENSOR NETWORKS", Springer- Verlog Berlin Heidelberg, 2010.

[2] F. Acer Saraci, " ARTIFICIAL INTELLIGENCE AND NEURAL NETWOKS", Springer-Verlag Berlin Heidelberg, 2006.

[3] Vemuri V. R.," ARTIFICIAL NEURAL NETWORK: Concepts and Control Applications", IEEE Computer Society Press, 1995.

[4] Bishop C. M.," NEURAL NETWORKS FOR PATTERN RECOGNITION", Oxford University, New York, 1995.

[5] David L.,"A BASIC INTRODUCTION TO FEEDFORWARD BACKPROPAGATION NEURAL NETWORKS", David Levering, 2009.

[6] JunS., Choi-HongL., Xiao-Jun W.," PARTICAL SWARM OPTIMIZATION: Classical and Quantum Perspectives", Taylor &Francis, LCC,2012.

[7] Maurice Clerc," PARTICAL SWARM OPTIMIZATION", ISTE Ltd., 2006.

[8] Li-Yeh C.,Yu-da Lin, Cheng-Hong Y.," AN IMPROVED PARTICLE SWARM OPTIMIZATION FOR DATA CLUSTERING", International Multi conference of Engineers and Computer Scientists, Vol.1,2012. https://www.iaeng.org/publication/IMECS 2012-pp 440-445.pdf.

[9] Maria G. A., Pierre D.," PARTICAL SWARM OPTIMIZATION (PSO): An Alternative Method for Composite Optimization", 10th World Congress on Structural and Multidisciplinary Optimization, 2013. https://www2.mae.nfl.edu/mdo/papers/5334.pdf.

[10] Te-Jen S., Ming-Yuan H., Yuei-Jyun S.," N ADAPTIVE PARTICLE SWARM OPTIMIZATION FOR THE COVERAGE OF WIRELESS SENSO R NETWORK", Springer-Verlag Berlin

Heidelberg, 2011.
https://www.mfile.narotama.ac.id.com.

[11] K. Kavitha, M. Mohamed S.," PARTICLE SWARM OPTIMIZATION BASED QoS AWARE ROUTING FOR WIRELESS SENSOR NETWORKS", International Journal for Scientific research & Development, Vol.2, Issue 07, 2014. https://www.ijsrd.com.

[12] Haiping H., Junqing Z., Ruchuan W., Yishing Q.," SENSOR NODE DEPLOYMENT IN WIRELESS SENSOR NETWORKS BASED ON IONIC BOND-DIRECTED PARTICLE SWARM OPTIMIZATION", Natural Sciences Publishing Cor., 2014. https://www.naturalpublishing.com.

[13] Mark H.B., Martin T. H., Haward B. D.," NEURAL NETWORK TOOLBOX: User Guide", The Math Works Inc, 2014.

IJSER