Simulation Design of an Artificial Neural Network for Sensor Network Applications

Hanan A. R. Akkar Ph.D (Prof.)* Aied K. AL-Samarrie Ph.D (Asst.Prof.)* Azzad Bader Saeed (Lecturer)*

Abstract

The aim of this study, is to design a simulation of Back-propagation Neural Network used for processing the data incoming from the sensor units of a sensor network (SN), and then presenting specific decisions. The Back-propagation Neural Network is one of the powerful Artificial Neural Network, cause it is a trained network, with optimization method of 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. *Traingda* function had been used as a training function for the proposed system, which is a gradient descent with adaptive learning rate 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 package, the testing process had offered stimulant results, whereas, the actual output had fitted the desired output, and the Mean Square Error had reached to zero with 87 iterations, where no previous research had reached to this optimal result for such design.

Keywords: Artificial Neural Network (ANN), Sensor Networks, SATLINS Function, Traingad Function.

* University of Technology

1. Introduction

Intelligent sensor networks are momentous systems at present time. This constructed from two important parts. The first part is the sensor unit, this unit consists of a sensor and an A/D (Analog-to-Digital) converter, the sensor used to translate the concentration of the effecting element to an analog voltage, while the analog-to-digital converter used for converting the sensor output voltage to a digital data. The second part is the intelligent simulation system, which may be saved in a processor system or in an FPGA (Field Programmable Gate Array) unit, it receives the digital data introduced by the sensor units and processes these data to produce a specific result [1][2].

One of the effective intelligent systems used in the intelligent sensor network is the *Back-propagation Neural Network System.*

Back-propagation 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[3][4].

Back-propagation 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. Back-propagation requires that the activation function used by the artificial neurons (or "nodes") be differentiable[5][6].

The whole Back-propagation process is intuitively very clear. When a learning pattern is clamped, the activation values are propagated to the output units, and the actual network output is compared with the desired output values, we usually end up with an error in each of the output units[7][8].

Let's call this error e_o for a particular output unit (o). We have to bring e_o to zero. The simplest method to do this is the greedy method: we strive to change the connections in the neural network in such a way that, next time around, the error e_o will be zero for this particular pattern. We know from the delta rule that, in order to reduce an error, we have to adapt its incoming weights according to[9]:

$$\Delta w_{ho} = (d_o - y_o)\, y_h \qquad \dots \qquad (1)$$

Where,

Δw_{ho} : Weigh change of the connections between output and hidden layers.

d_o : Desired output value at the output layer.
y_o : Actual output value at the output layer .
y_h : Actual output value at the hidden layer

That's step one. But it alone is not enough, when we only apply this rule, the weights from input to hidden units are never changed, and we do not have the full representational power of the feed-forward network as promised by the universal approximation theorem. In order to adapt the weights from input to hidden units, we again want to apply the delta rule. In this case, however, we do not have a value for δ for the hidden units. This is solved by the chain rule which does the following, distributes the error of an output unit (*o*) to all the hidden units that is it connected to, weighted by this connection. Differently put, a hidden unit (*h)* receives a delta from each output unit (*o*) equal to the delta of that output unit weighted with (multiplied by) the weight of the connection between those units.

The application of the generalized delta rule thus involves two phases: During the first phase the input (*x*) is presented and propagated forward through the network to compute the output values (y_o^p) for each output unit. This output is compared with its desired value d_o, resulting in an error signal (δ_o^p) for each output unit. The second phase involves a backward pass through the network during which the error signal is passed to each unit in the network and appropriate weight changes are calculated.

The weight of a connection is adjusted by an amount proportional to the product of an error signal δ, on the unit *k* receiving the input and the output of the unit *j* sending this signal along the connection:

$$\Delta_p w_{jk} = \gamma \delta_k^p y_j^p \qquad \dots \quad (2)$$

Where,
Δw_{jk} : Weight change of input connection of the p^{th} unit between the (j) sending layer and (k) receiving layer.
γ : Learning rate.
δ_k^p :Delta function of the p^{th} unit of the (*k*) receiving layer.
y_j^p : Actual output of the p^{th} unit of the (*j*) sending layer.

If the unit is an output unit, the error signal is given by:

$$\delta_o^p = \left(d_o^p - y_o^p \right) F'(s_o^p) \quad \ldots \quad (3)$$

Where,
δ_o^p : Delta function of p^{th} unit of the output layer.
d_o^p : Desired output of p^{th} unit of the output layer.
y_o^p : Actual output of p^{th} unit of the output layer.
$F'(s_o^p)$: Differentiation of the activation function of p^{th} unit of the output layer.

Take as the activation function *F* the 'sigmoid' function as defined:

$$y^p = F(s^p) = \frac{1}{1+e^{-s^p}} \quad \ldots \quad (4)$$

In this case the derivative is equal to:

$$F'(s^p) = y^p(1 - y^p) \quad \ldots \quad (5)$$

In this case the derivative is equal to such that the error signal for an output unit can be written as[9]:

$$\delta_o^p = \left(d_o^p - y_o^p\right) y_o^p \left(1 - y_o^p\right) \quad \ldots \quad (6)$$

Finally, the error signal for a hidden unit is determined recursively in terms of error signals of the units to which it directly connects and the weights of those connections. This is estimated by the following expression[9]:

$$\begin{aligned} \delta_h^p &= F'\left(s_h^p\right) \sum_{o=1}^{N_o} \delta_o^p w_{ho} \\ &= y_h^p \left(1 - y_h^p\right) \sum_{o=1}^{N_o} \delta_o^p w_{ho} \end{aligned} \quad \ldots \quad (7)$$

Where,
δ_h^p : Delta function of p^{th} unit of the hidden layer.
$F'(s_h^p)$: Differentiation of the activation function of p^{th} unit of the hidden layer.
w_h^o : Weight connection between hidden and output layer.
y_h^p : Actual output of p^{th} unit of the hidden layer.
This expression for the sigmoid activation function.

2. Previous Related Works

(K. Henkel and D. Schmeiber , 2002) had solved the delicate problem of independently determining the concentration of carbon dioxide and relative humidity in a gas mixture by recording the signals of two quartz microbalances coated with a functionalized sensitive polymer layer. The data were analyzed by Back-propagation – based neural networks, and they had tested two different architectures, a one-stage net and a two stage net (which consist of two network structures in series) with respect to their generalization ability. They didn't implement their proposed system in an FPGA cause they had used non-linear activation functions for the hidden and output layers.

(Muhammed K., B. Muhammed K. and T. Muhammed J., 2012) had described the implementation of the Back-propagation algorithm for use in Wireless Sensor Network (*WSN*) application. This application had been performed over an FPGA and MATLAB software package. This paper studies the architecture of a neural wireless sensor network designed to identify technical conditions (temperature, humidity, and light, etc.) of the base station of wireless sensor networks. This work had used A/D converter ICs and microcontrollers in the sensor units.

(Guo W. and Juan W., 2014) had presented a design and implementation of wireless sensor network nodes based on Back-propagation neural network. The proposed Back-propagation neural network has been trained by training parameters of Routing protocol for wireless mobile sensor networks, which can be active, or can be passive depending on the proposed design. This work wasn't implemented in FPGA, cause it had used non-linear activation function for the proposed neural network.

3. 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), inputs (a) and (b) are the least and most significant bits of the sensor (1) output data, while inputs (c) and (d) are the least and most significant bits of the sensor (2) output data. Outputs (x), (y), and (z) are the desired output data of the proposed system. For example, if the input data introduced by sensor unit (1) is (ba)= (01) (LOW) state, and the input data introduced by sensor unit (2) is (dc)= (01) (LOW) state too, then the proposed system will produce an output data (zyx)=(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.

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

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 (Traingda) had been chosen as a learning function for the proposed system, it means gradient descent with adaptive learning rate function, used for updating the weights and biases (where the biases are included in the weights blocks) input lines of the hidden and output layers, which is fast training method.

From repetitive training of the proposed system for reaching to the accurate results, the value 0.05 had been chosen for the learning rate, and the value 300 had been chosen for the maximum training Epochs.

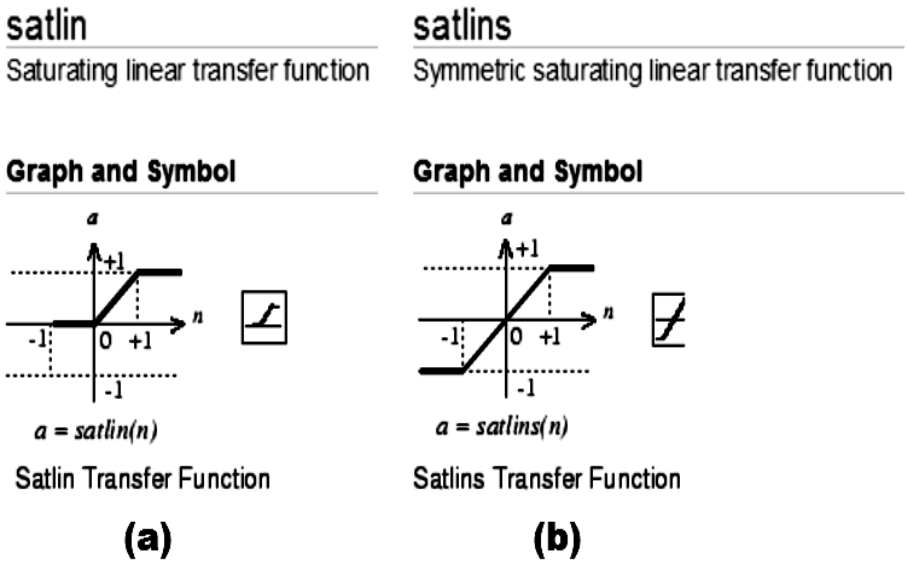

Fig.(1): Linear activation functions[13]: SATLINfunction.
a- SATLINS function.
b- SATLINS function.

Finally, the previous considerations had helped the programmer to write 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 de-multiplexer 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 learning 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, and 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.

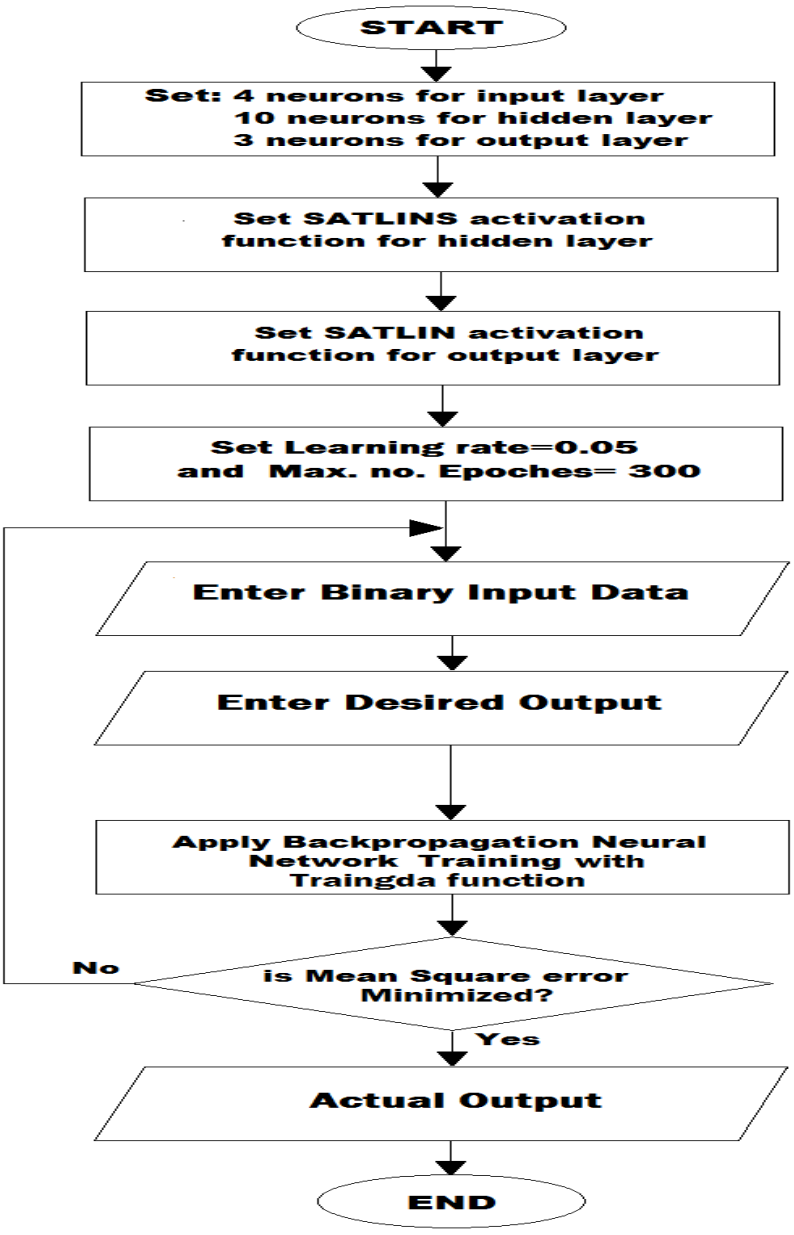

Fig.(2): Flowchart of training the proposed system.

4. Results and discussion

After executing the proposed MATLAB software, a simulation system block will be generated, this block has four inputs 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 de-multiplexer 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.

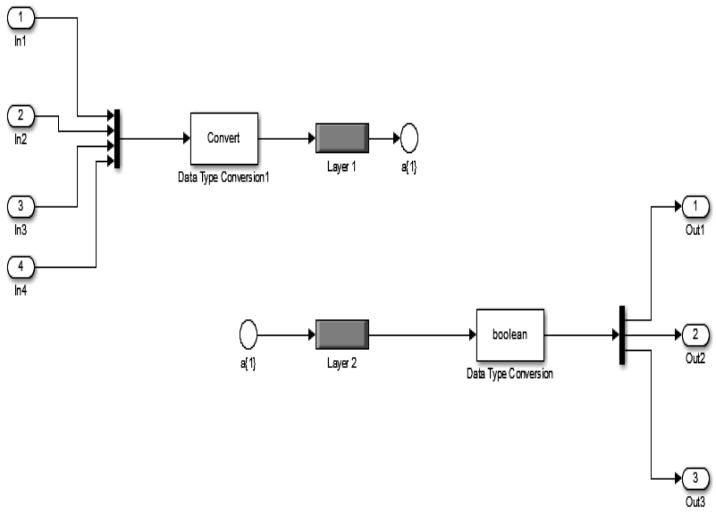

Fig.(3): Internal network of the proposed system block.

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 thin 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.

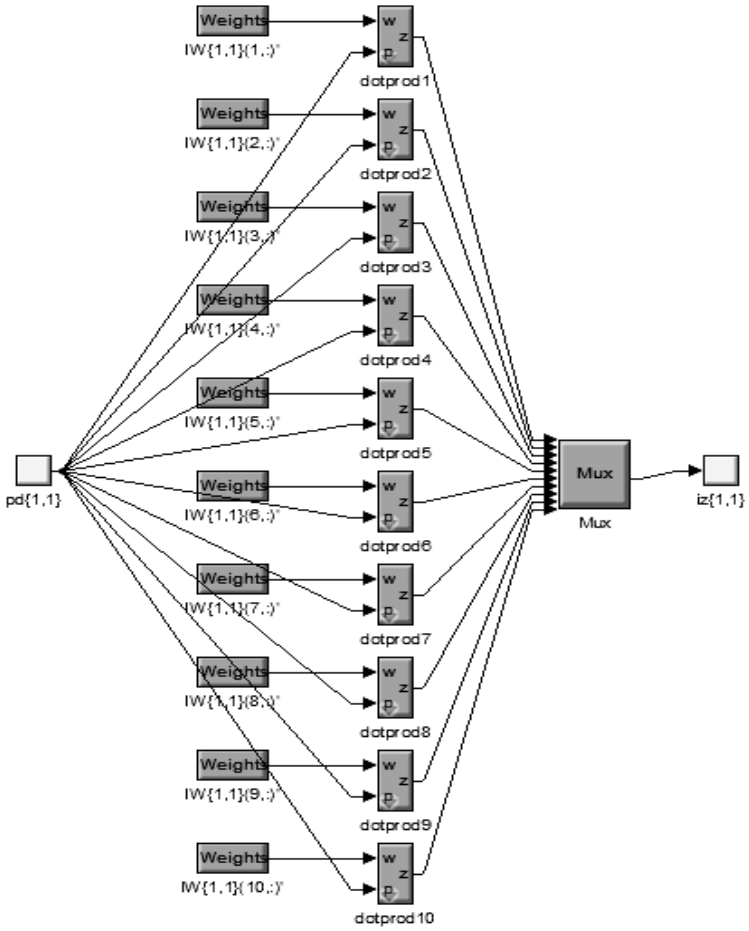

Fig.(4): Internal network of weight block of layer(1) block.

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.

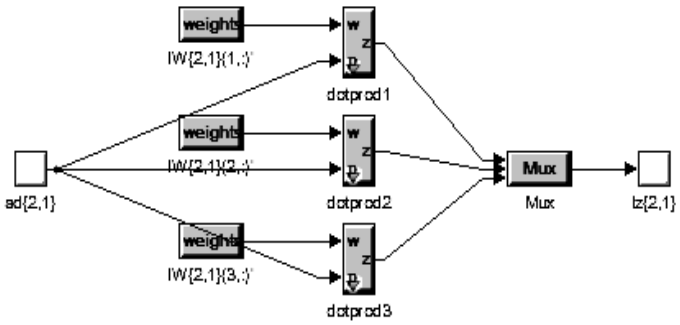

Fig.(5): Internal network of weight block of layer(2) block.

The data type of the overall simulation block must be converted from (double) data type to fixed data (fixdt) type (double means Double-Precision Floating point data which consists of 64 bits, bit 63 for sign, bits 62-52 for exponent, bits 51-0 for fraction of 1, while the fixdt means fixed-point or floating point data, the no. of bits of the sign and beyond and before the point are selectable), 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.

After executing the simulation program, a results window will be displayed as shown in Figure(6), this window shows the important results of the performance (which represents the relation between mean square error vs the no. iteration) and gradient of the simulation system block, one can see from this figure that the gradient starts from value 0.61 at epoch 0 and finishes with value zero at epoch 87, and the mean square error starts from value 0.449 at epoch 0 and finishes with value zero at epoch 87, which they are accurate results.

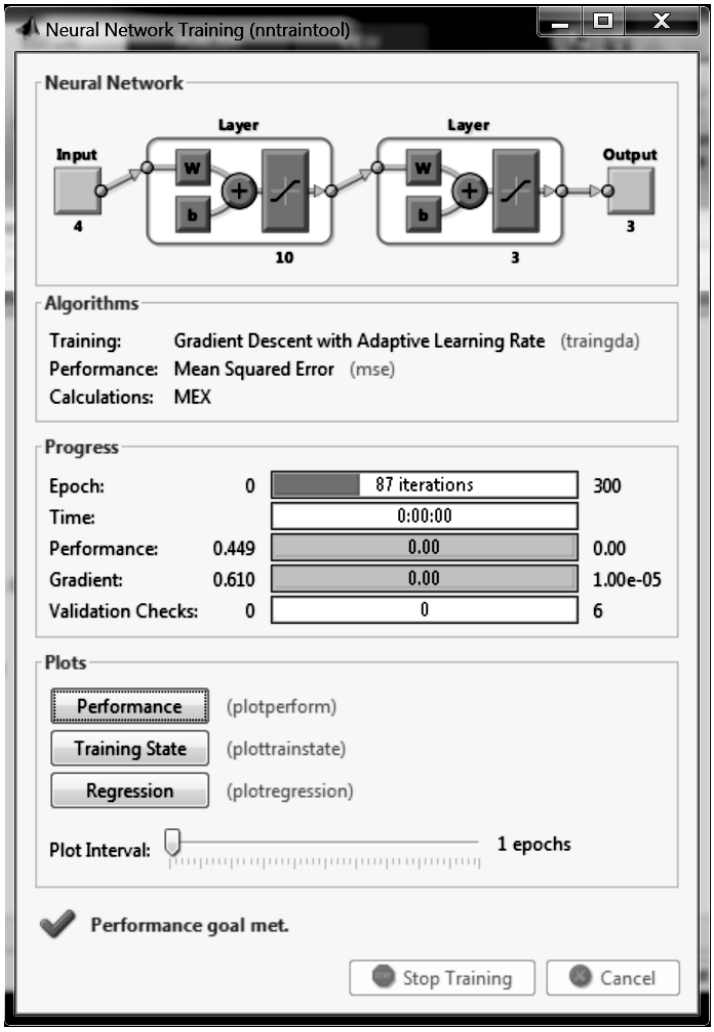

Fig.(6): primary results of the proposed system.

After clicking on the performance option, a curve line will be exhibited, as shown in Figure (7) , one can see from this figure that the mean square error gradually decreases in continuous manner to reaches the zero value, which leads to the successful learning with time proceeding.

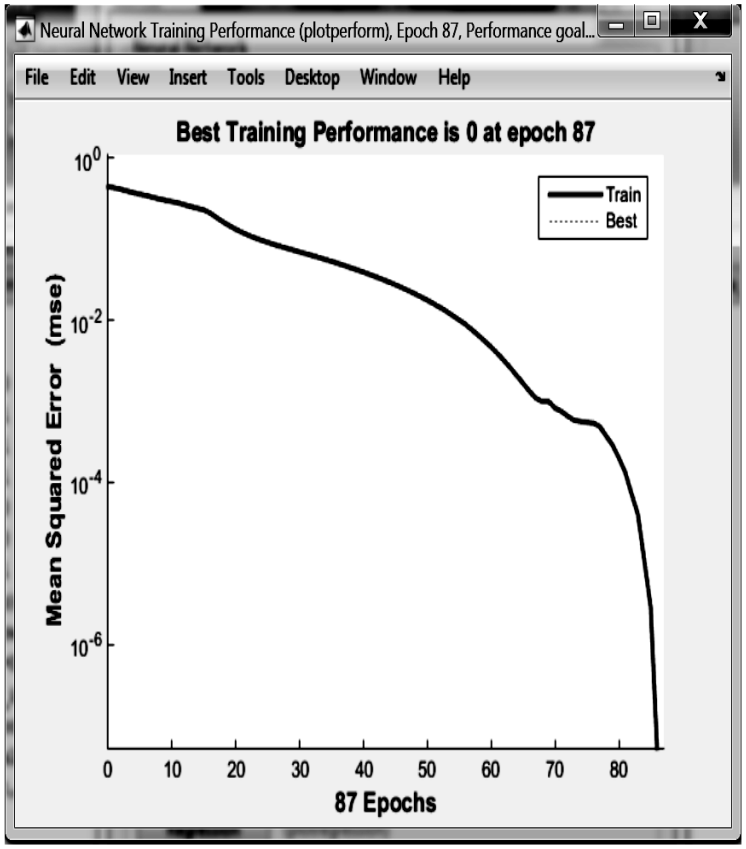

Fig.(7): Performance result window of the proposed system.

After clicking on the regression option, a linear line will be exhibited as shown in Figure(8), which shows the dashed line that identical to the continues line, where, the dashed line represents the equality of the actual output with the desired (Target) output, while, the continuous line represents the relation between the desired output and the actual output.

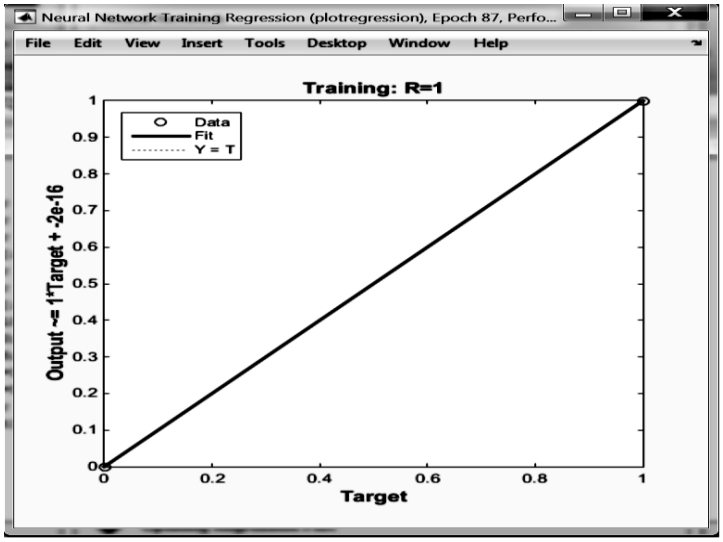

Fig.(8): Regression output of the proposed system.

From this result, one can see that the actual (real) output is identical to the desired output. One can conclude from the experimental work, if the number of hidden layers had increased or the number of neurons of the single hidden layer had decreased, the accuracy of the results had decreased as shown in Figure (9).

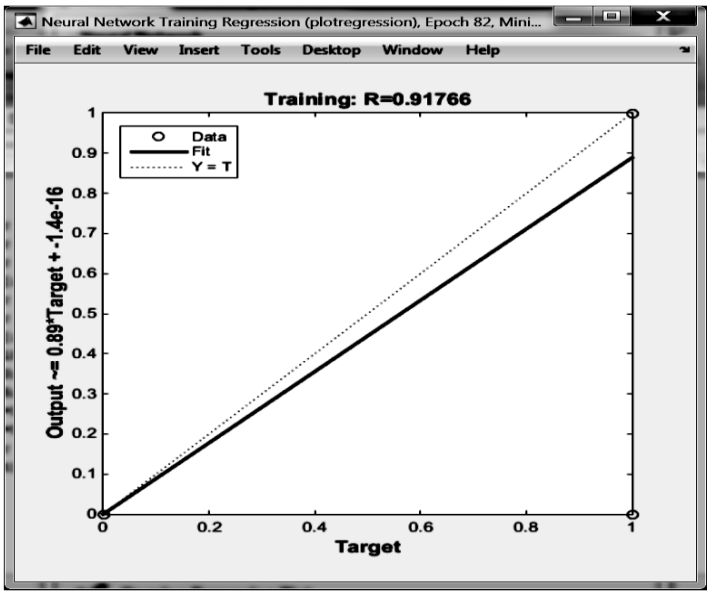

Fig.(9): Regression output of a results with less accuracy.

One can see from this figure that dashed line is not identical to the continuous line, i.e. the actual output data is not fitted to the desired output data.

The state diagram of the proposed system shown in Figure (10), it exhibits the relationship between states of the four input signal (a,b,c,d), and states of the three output signals (x,y,z).

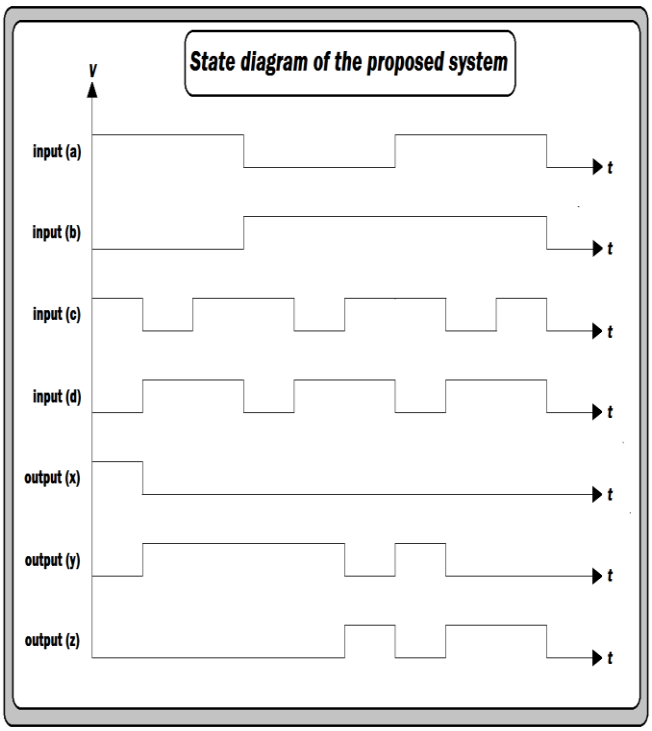

Fig. (10): The state diagram of the proposed system.

(K. Henkel and D.Schmeiber, 2002) had proposed a Back-propagation Neural Network as an intelligent system for a sensor network, that had used non-linear activation function for the hidden and output layers, and had used *TRAINGD* (Gradient Descent) training function for training the network, and also they had suggested two networks, the first is a single-stage network, and the second is a two-stage network. The performance results for this work had been presented in Figures (11),(12). One can see from these figures that the mean square error had reached to (0.117934) at epoch 300 for the single-stage network, while the mean square error for the two-stage had reached to (0.0140504) at epoch 300, i.e. the mean square error of the two-stage network had improved, and one can see from these results that the mean square error is so more than the mean square error of our work that shown in Figure (7).

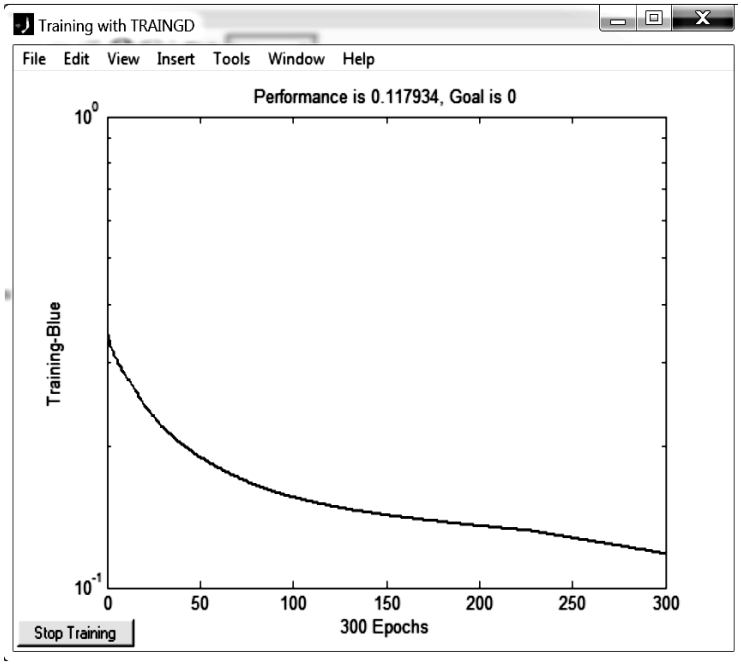

Fig.(11): Performance result of the single-stage BB neural network of related work [10]

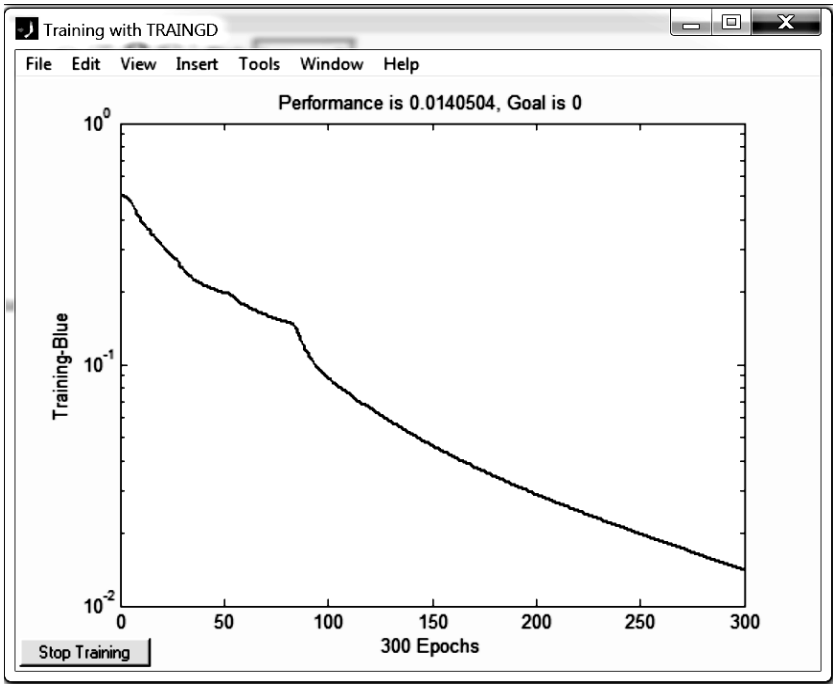

Fig.(12): Performance result of the two-stage BB neural network of related work [10]

(Muhammed K. and B. Muhammed, 2012) had proposed a Back-propagation neural network for a wireless sensor network, which had used *TRAINLM* (Levenberg-Marquardt) training function for learning the network, and had used linear activation functions for the hidden and output layers. the performance result of this work had been presented in Figure(13), one can see from this figure that the mean square error of this work had reached to (3.53527×10^{-30}) at epoch 16, which is more than the result of our work.

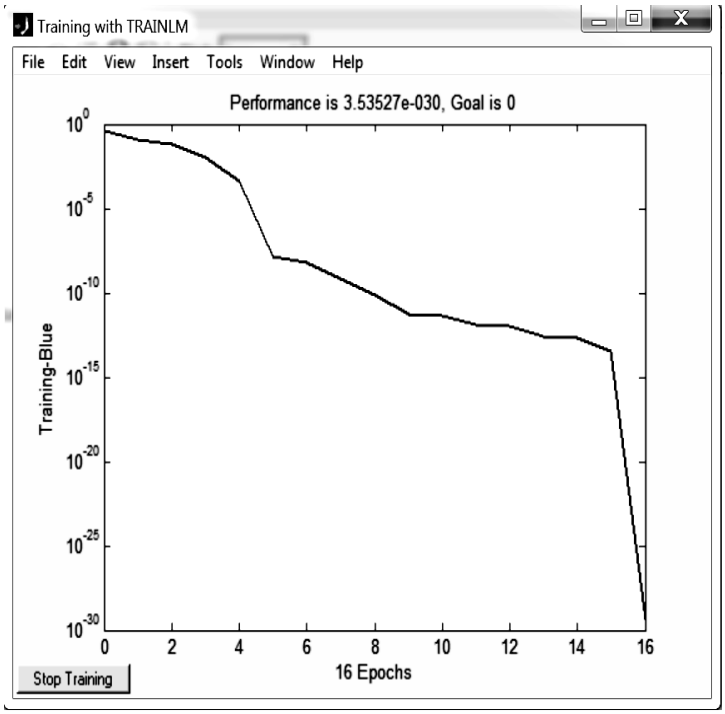

Fig.(13): Performance result of the BB neural network of related work [11]

Guo W. and Juan W., 2014) had proposed a Back-propagation neural network for a sensor network that used non-linear activation functions for the hidden and output layers, and had used *TRAINGDX* (Gradient Descent with Momentum and Adaptive Learning Rate) training function for learning the network. the performance result had been shown in Figure(14), one can see from this figure that the mean square error had reached to (4.41471×10^{-12}) at epoch 190, which is more than the mean square error of our result that shown in Figure(7).

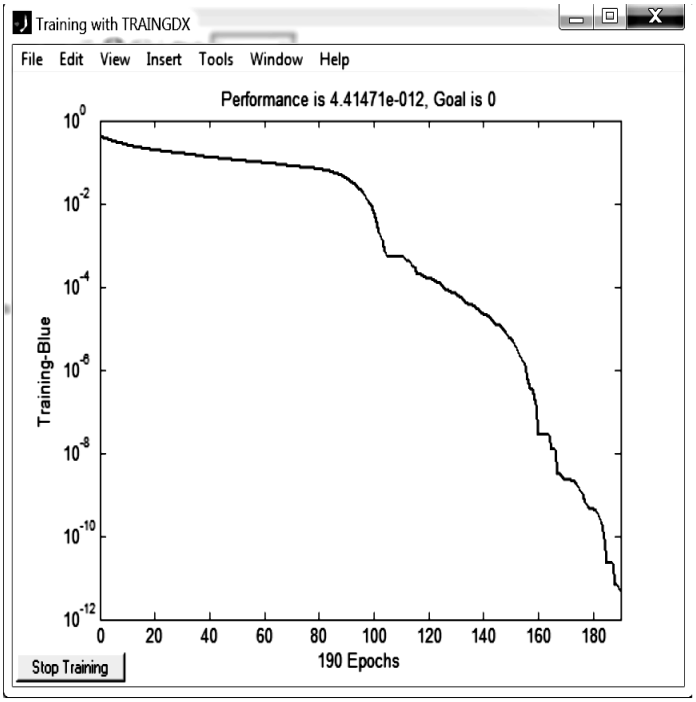

Fig.(14): Performance result of the BB neural network of related work [12]

5. Conclusions

The sensor units can be increased by increasing the input lines of the proposed system, which leads to increasing the neurons of the input layer, therefore, the size of the proposed simulation software will increase (i.e. we must choose the suitable FPGA kit that support this software) because the no. of multipliers, adders and multiplexers, etc. will be increased .

Practically, non-linear activation functions cannot be used with the proposed system, because the MATLAB package cannot convert this system to VHDL (VHSIC (Very high-Speed Integrated Circuit) Hardware Design Language) code program, and then it cannot be downloaded into an FPGA .

One can suggests more output levels to increase the accuracy and the resolution of the system output, which leads to increasing the neurons of the output layer, and this process is also increased the size of the system software.

Using Traingda training function, one can get accurate results with zero mean square error and fast training process to update the weights and biases of inputs of the hidden and output layers.

By practical multi-attempt to execute the proposed simulation software, one can reach to this fact, the Increasing of the hidden layers, or decreasing the neurons of the single hidden layer, leads to decreasing the accuracy of the results.

References

[1] Anna Hac," WIRELESS SENSOR NETWORK DESIGNS", John Wiley & Sons Ltd, England, 2003.

[2] Utz R., Cormac J. S.,"WIRELESS SENSOR NETWORKS", Springer- Verlog Berlin Heidelberg, 2009.

[3] Duda R. O., Hart P. E., Stroke D. G.," PATTERN RECOGNITION", Wiley Inc., 2001.

[4] Paul J. Werbos," THE ROOTS O BACKPROPAGATION: From Ordered Derivatives to Neural Networks and Political Forecasting", John Willy & Sons, Inc., 1994.

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

[6] L. R. Medsker, L. C. Jain," RECURRENT NEURAL NETWORK: Design and Application", CRC Press LLC. 2001.

[7] F. Acar Saraci,"ARTIFICIALINTELLIGENCE AND NEURAL NETWOKS", Springer - Verlag Berlin Heidelberg, 2006.

[8] Daniel S., Ian C., Daming S., Wing W.," SENSITIVITY ANALYSIS FOR NEURAL NETWORKS", Springer-Verlog Heidelberg, 2010.

[9] David L.,"A BASIC INTRODUCTION TO FEEDFORWARD BACK-PROPAGATION NEURAL NETWORKS", David Leverington, 2009.

[10] K. Henkel, D. Schmeiber, "BACK- PROPAGATION - BASED NEURAL WITH A TWO SENSORS SYSTEM FOR MONITORING CARBON DIOXIDE AND RELATIVE HUMIDITY", Springer - Verlag, ISSN: 1618-1650, 2002.

[11] Muhammed K., B. Muhammed K., T. Muhammed J., "FPGA BASED EURAL WIRELESS SENSOR NETWORK", The 13th International Arab Conference on Information Technology, 2012.

[12] Guo W., Juan W., " DESIGN AND IMPLEMENTATION OF WIRELESS SENSOR NETWORK NODES BASED ON BP NEURAL NETWORK", Journal of chemical and Pharmaceutical Research, ISSN: 0975- 7384, 2014.

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

تصميم محاكاة شبكة عصبية ذكية لتطبيقات شبكة الحساسات

أ.د. حنان عبدالرضا عكار* أ.م.د. عايد خلف السامرائي* م. ازاد بدر*

المستخلص

الهَدف مِن هذا البَحث هوَ لِتَصميم مُحاكاة لِشكبة عَصبية نَوع الإنتشار الخَلفي مُستخدمة لِمُعالجة البَيانات القادِمة من وحدات الحَساسات لِشبكة حَساسات, و كذلك لِتقديم قَرارات مُحددة. ألشبكة العَصبية نوع الإنتشار الخَلفي هي إحدى الشبكات الذكية الفَعّالة لأنها تعتبر شكبة مُدرَّبة مع طَريقة مُثلى لِتحديث الأوزان والإنحيازات لِطبقات الخَفية والخَرج.

قد أُستُخدِمت الدَوال SATLINS و SATLIN كَدَوال تَنشيط خَطّية لِطبقات الخَفية والخَرج. و قد أستُخدِمت الدالة Traingda لتدريب الشبكة العصبية المُقترحة, وهي طريقة تدريب بتقليل الإنحدار مع مُعدّل تعليم مُتكيف. ومِن الجَدير بالذكر بأنهُ لايوجد بحث سابق قد استخدم هذه الدوال المذكورة سَوِيَّةً في هكذا تحليلات.

هذه المنظومة قَد تَمَّ مُحاكاتها و اختبارها باستخدام الحُزمة البرمجية MATLAB, و قد أظهرت نتائج مُؤثرة في النفس, حيث ان الخروج الحقيقي قد طابق الخَرج المرغوب, وان نسبة الخطأ قد وصلت الى قيمة الصفر ب 87 محاولة تدريب, حيث لايوجد بحث سابق قد وصل الى هذه النتيجة المُثلى.

*الجامعة التكنولوجية