# Nuclear Research Reactors Accidents Pattern Recognition Using Artificial Neural Networks

** Abdelfattah A. Ahmed; *Nwal Ahmed Alfishawy; * Mohamed A. Albrdini, and **Imbaby I. Mahmoud

* Dept of Comp. Sci. & Eng., Faculty of Electronic Engineering, Minufiya University, Minuf, Egypt.
** Atomic Energy Authority, Nuclear Research Center, Inshas, Egypt.
fatt231153@yahoo.com

**Abstract**: The patterns recognition of measured quantities for the diagnostic purposes in the field of nuclear research reactors is very important. It represents one of the fundamental tasks for the operation and accidents management. In this paper, the Nuclear Research Reactors accident's pattern recognition is tackled within neural network approach. Such patterns are introduced initially without noise. The simulated output values of the matrix's diagonal are larger than 0.9, (approximately equal 1), this means the outputs is approximately equal the targets and the network is well trained. To increase the reliability of such neural network, the noise ratio up to 50% was added for training in order to ensure the recognition of these patterns if it introduced with noise. Also, because of the limited amount of data (patterns), this work has taken care to increase the size of these data (patterns) when it introduced as training packages, by adding different random noise ratios as different sets at different times to ensure proper training of the neural network components. The neural network has been tested after training, and also finally tested by providing separate data patterns to ensure the ability of the constructed network to recognize these patterns. Experiments have shown excellent results; where the network did not make any errors for input vectors (patterns) with the noise level from 0.00 up to 0.14. When the noise level is larger than 0.15 was added to the input vectors (patterns) both networks began making errors.

## 1. Introduction

As a basic definition, a neural network is an assembly of interconnected processing elements used to represent a real world system, i.e. a neural network is an interconnected web of individual neurons. In simpler terms, a neural network is a mathematical system used to approximate a system output based on a specific input. Or, it is a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experiential knowledge and making it available for use [6].

Matlab's Neural Network Toolbox is software that provides comprehensive support for many proven network paradigms, as well as graphical user interfaces (GUIs) that enable to design and manage neural networks. The Toolbox supports both supervised and unsupervised neural networks. The supervised neural networks are trained to produce desired outputs in response to sample inputs, making them, particularly well suited to modeling and controlling dynamic systems, recognizing noisy data, and predicting a future event, which is our domain, demand [5-8].

In this work, a neural network is designed and trained to recognize the 9 accidents of the nuclear reactors. By the aid of reactor operation crew and Safety Analysis Report (SAR) of the reactor, also the Atomic Energy Authority (AEA) experts, data sets were collected for the eight accidental cases listed below plus the normal operation case (Classes) as shown in Figure (1). So the total cases, which we have, are nine. The result is that each accident is represented as a 3-by-5 grid of Boolean values.

| TRO | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|-----|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TR1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| TR2 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| TR3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| TR4 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| TR5 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| TR6 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| TR7 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| TR8 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Figure (1)**:** Sample of reactor accidents data patterns.

However, the data sets perhaps are not perfect, and the accidents can suffer from noise. Perfect recognition of ideal input vectors is required and reasonably accurate recognition of noisy vectors, see Figure (2).

The nine 15-element input vectors are defined in the function *accmodels* as a matrix of input vectors called *accidents*. The target vectors are also defined in this file with a variable called, *targets*. Each input vector is a 15-element vector (3-by-5), Figure (3), with a 1 in the position of the accident it represents, and 0's everywhere else. For example, the TR0 is to be represented by a 1 in the first element (as TR0 is the first accident of the accidents), and 0's in elements two through fifteen.
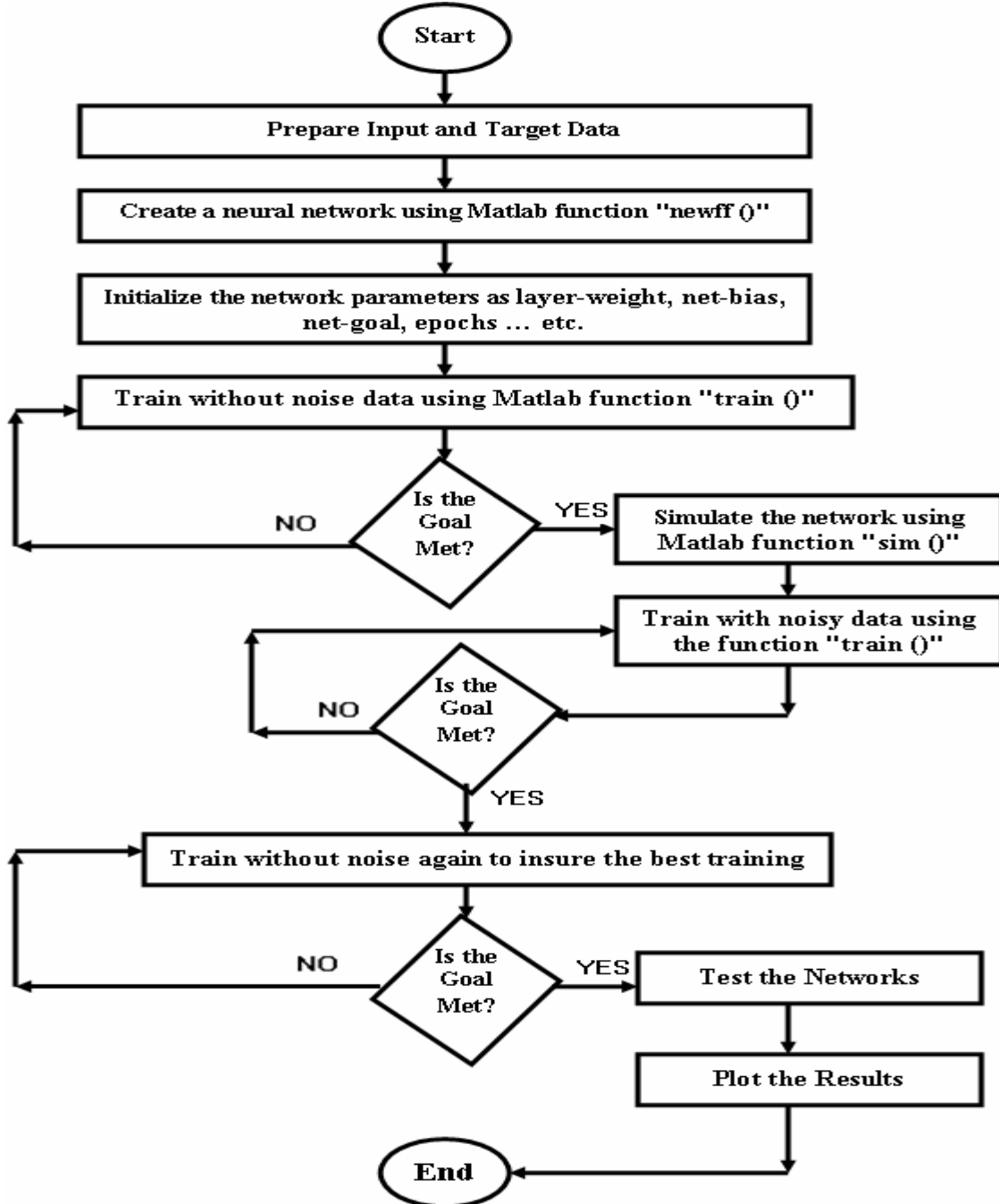


Figure (2): Program flowchart using Matlab.

$$TR0 = [0\ 0\ 0\ ...\qquad .....\qquad TR8 = [1\ 1\ 1\ ...$$
$$0\ 1\ 1\ ...\qquad .....\qquad\qquad 1\ 1\ 1\ ...$$
$$1\ 1\ 1\ ...\qquad .....\qquad\qquad 1\ 1\ 1\ ...$$
$$1\ 1\ 1\ ...\qquad .....\qquad\qquad 1\ 1\ 1\ ...$$
$$1\ 1\ 1]';\qquad .....\qquad\qquad 1\ 1\ 1]';$$

Figure (3): Samples of (15x9) matrix of (3x5) bit maps for each Accident.

This paper comprises five sections. After this introductory section, Neural Network Implementation is in section (2). Results and Discussions are in section (3). Conclusion is in section (4). Finally, References are in section (5).

## 2. Neural Network Implementation

The network receives the 15 Boolean values as a 15-element input vector (3-by-5). It is then required to identify the accident by responding with a 9-element output vector. The 9 elements of the output vector each represent an accident.

To operate correctly, the network should respond with a 1 in the position of the accident being presented to the network. All other values in the output vector should be 0.

In addition, the network should be able to handle noise. In practice, the network does not receive a perfect Boolean vector as input. Specifically, the network should make as few mistakes as possible when recognizing vectors with noise of mean 0 and standard deviation of 0.2 or less.

### 2.1 Network Architecture

The neural network needs 15 inputs and 9 neurons in its output layer to identify the accidents. The network is a two-layer log-sigmoid/log-sigmoid network. The log-sigmoid transfer function was picked because its output range (0 to 1) is perfect for learning to output Boolean values.
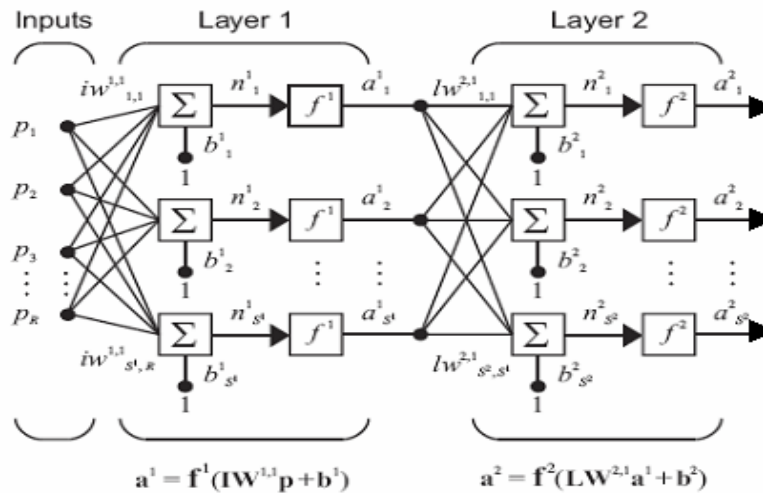


$$a^1 = f^1(IW^{1,1}p + b^1)\qquad\qquad a^2 = f^2(LW^{2,1}a^1 + b^2)$$

Figure (4): neural network design for accidents patterns recognition

$a^2 = f^2 (LW^{2,1} (f^1 (LW^{1,1}p + b^1) + b^2) = y_j,$ --- (1)
where: j is neurons in the output layer,

The hidden (first) layer has 10 neurons. This number was picked by guesswork and experience. If the network has trouble learning, then neurons can be added to this layer, Figure (4).

The network is trained to output a 1 in the correct position of the output vector and to fill the rest of the output vector with 0's. However, noisy input vectors can result in the network's not creating perfect 1's and 0's. After the network is trained the output is passed through the competitive transfer function *compet*. This makes sure that the output corresponding to the accident most like the noisy input vector takes on

a value of 1, and all others have a value of 0. The result of this post-processing is the output that is actually used.

### 2.2 Network Training

To create a network that can handle noisy input vectors, it is best to train the network on both ideal and noisy vectors. To do this, the network is first trained on ideal vectors until it has a low sum squared error. Then the network is trained on 10 sets of ideal and noisy vectors. The network is trained on two copies of the noise-free accidents at the same time as it is trained on noisy vectors. The two copies of the noise-free accidents are used to maintain the network's ability to recognize ideal input vectors.

Unfortunately, after the training described above the network might have learned to recognize some difficult noisy vectors at the expense of properly recognizing a noise-free vector. Therefore, the network is again trained on just ideal vectors. This ensures that the network respond perfectly when presented with an ideal accident. All training is done using backpropagation with both adaptive learning rate and momentum, with the function *'traingdx'*. The Error (Performance) is calculated in terms of Sum-Squared Error (*SSE*), equation (2), or Mean-Squared Error (*MSE*), equation (3), according to the choice of the performance function at the network creation by the following two equations:

$$SSE = \sum_{i=1}^{N} \sum_{K=1}^{K} (t_{i,k} - y_{i,k})^2 \qquad \text{--- (2)}$$

$$MSE = \frac{1}{N\,K} \sum_{i=1}^{N} \sum_{k=1}^{K} (t_{i,k} - y_{i,k})^2 \text{ ---}$$

(3)

Where *N* and *K* denote the number of patterns and output nodes used in the training respectively, *i* denotes the index of the input pattern (vector), *k* denotes the index of the output node, $t_{i,k}$ and $y_{i,k}$ express the desired output (target) and actual output values of the $k^{th}$ output

node at $i^{th}$ input pattern, respectively. The calculation of the output is according to figure (4) for two layers network using equation (1).

### 2.2.1 Training without Noise

The network is initially trained without noise for a maximum of 5000 epochs or until the network sum squared error falls beneath 0.1. Figure (5) shows the output every 20 epochs, and the training stop when the Performance goal is met at epoch number158. The procedure for backpropagation training is as follows, (for each input vector associate a target output vector):

while not STOP
 STOP=TRUE
  for each input vector
   perform a forward sweep to find the actual output
   obtain an error vector by comparing the actual and target output
    if the actual output is not within tolerance set STOP= FALSE
   perform a backward sweep of the error vector
   use the backward sweep to determine weight changes
   update weights
 end while



```
traingdx-calcgrad, Epoch 0/5000,    SSE 19.9605/0.01,    Gradient 8.02798/1e-006
traingdx-calcgrad, Epoch 20/5000,   SSE 10.6245/0.01,    Gradient 3.34315/1e-006
traingdx-calcgrad, Epoch 40/5000,   SSE 7.82391/0.01,    Gradient 0.770611/1e-006
traingdx-calcgrad, Epoch 60/5000,   SSE 7.32363/0.01,    Gradient 0.701802/1e-006
traingdx-calcgrad, Epoch 80/5000,   SSE 6.0438/0.01,     Gradient 0.673944/1e-006
traingdx-calcgrad, Epoch 100/5000,  SSE 3.62131/0.01,    Gradient 0.528835/1e-006
traingdx-calcgrad, Epoch 120/5000,  SSE 1.02062/0.01,    Gradient 0.29785/1e-006
traingdx-calcgrad, Epoch 140/5000,  SSE 0.0620723/0.01,  Gradient 0.0359726/1e-006
traingdx-calcgrad, Epoch 158/5000,  SSE 0.00994019/0.01, Gradient 0.0062113/1e-006

traingdx, Performance goal met.
```

Figure (5): neural network training output

Figure (6) show the plot of performance vs. number of epochs. Figure (7) show the network response simulation. Figure (7-a) show a 9x9 simulation output matrix's diagonal values that
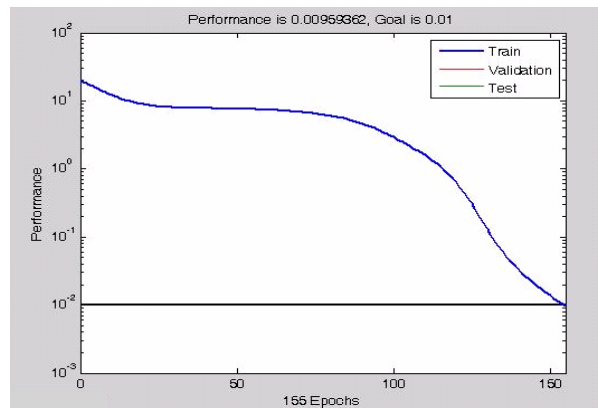


Figure (6): Performance vs. Number of Epoch

are larger than 0.9, and Figure (7-b) show a 9x9 matrix with diagonal values equal 1, which means that the outputs is equal the targets and the network is trained well.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0.9842 | 0.0139 | 0.0007 | 0.0096 | 0.0003 | 0.0083 | 0.0000 | 0.0086 | 0.0056 |
| 0.0095 | 0.9784 | 0.0048 | 0.0001 | 0.0004 | 0.0139 | 0.0022 | 0.0002 | 0.0133 |
| 0.0003 | 0.0072 | 0.9755 | 0.0018 | 0.0014 | 0.0004 | 0.0051 | 0.0100 | 0.0144 |
| 0.0180 | 0.0000 | 0.0005 | 0.9673 | 0.0087 | 0.0003 | 0.0062 | 0.0160 | 0.0083 |
| 0.0009 | 0.0016 | 0.0011 | 0.0005 | 0.9793 | 0.0204 | 0.0109 | 0.0006 | 0.0061 |
| 0.0033 | 0.0181 | 0.0005 | 0.0000 | 0.0098 | 0.9744 | 0.0002 | 0.0055 | 0.0001 |
| 0.0000 | 0.0002 | 0.0002 | 0.0004 | 0.0032 | 0.0001 | 0.9748 | 0.0162 | 0.0087 |
| 0.0120 | 0.0004 | 0.0122 | 0.0068 | 0.0009 | 0.0139 | 0.0191 | 0.9767 | 0.0001 |
| 0.0031 | 0.0057 | 0.0090 | 0.0173 | 0.0065 | 0.0000 | 0.0130 | 0.0000 | 0.9836 |

Figures (7-a)

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Figures (7-b)

Figure (7):  neural network output for Accidents Patterns Recognition

### 2.2.2 Training with Noise

To obtain a network not sensitive to noise, a new copy of the neural network was made and trains this network with two ideal copies and two noisy copies of the vectors of accidents. The target vectors consist also of four copies of the vectors in target. The noisy vectors have noise of mean 0.1 and 0.2 added to them. This forces the neuron to learn how to properly identify noisy accidents, while requiring that it can still respond well to ideal vectors.

$P = [(P + randn(R , Q)* \text{ noise percent } \%)]$

To train with noise, the maximum number of epochs is reduced to 300 and the error goal is increased to 0.6, reflecting that higher error is expected because more vectors (including some with noise), are being presented.

### 2.3.2.1 Results of training with noise

An example of the ten passes, for training the network with noise shown in the following paragraphs. In pass2, as example, the goal is met after 5 epochs, where it means one of the stopping criteria is met. The stopping criterion here is the sum of squares errors (SSE).

**Pass = 2**
traingdx-calcgrad, Epoch 0/300, SSE 0.67199/0.6, Gradient 1.35261/1e-006          (4)
traingdx-calcgrad, Epoch 5/300, SSE 0.588967/0.6, Gradient 1.10811/1e-006
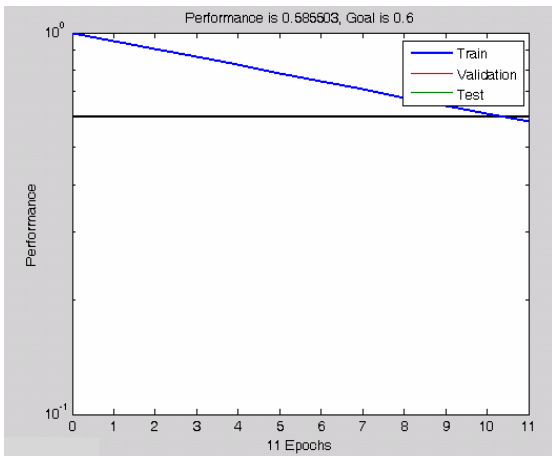traingdx, Performance goal met, Figure (8).



Figure (8): training with noise (Pass2)

**Pass = 4**
traingdx-calcgrad, Epoch 0/300, SSE 0.998751/0.6, Gradient 2.18894/1e-006
traingdx-calcgrad, Epoch 11/300, SSE 0.585503/0.6, Gradient 1.29207/1e-006
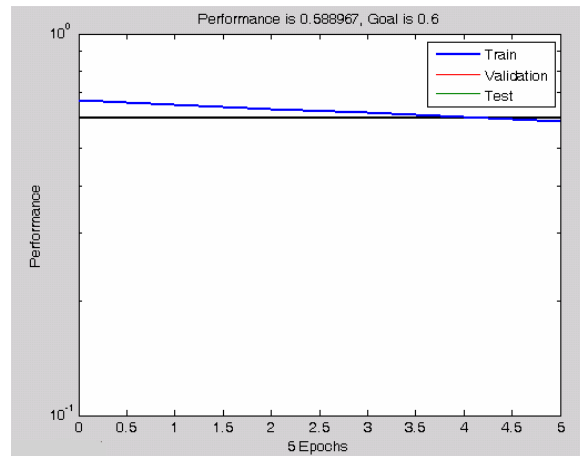traingdx, Performance goal met, Figure (9).



Figure (9): training with noise (Pass4)

**Pass = 7**
traingdx-calcgrad, Epoch 0/300, SSE 1.22755/0.6, Gradient 1.04305/1e-006
traingdx-calcgrad, Epoch 20/300, SSE 0.912542/0.6, Gradient 1.58917/1e-006
traingdx-calcgrad, Epoch 29/300, SSE 0.592562/0.6, Gradient 1.11897/1e-006
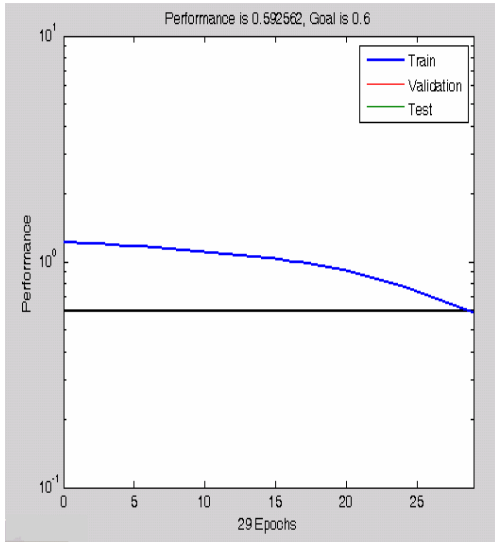traingdx, Performance goal met, Figure (10).

Figure (10): training with noise (Pass7)

**Pass = 9**
traingdx-calcgrad, Epoch 0/300, SSE 0.752496/0.6, Gradient 2.33902/1e-006
traingdx-calcgrad, Epoch 4/300, SSE 0.567219/0.6, Gradient 1.64352/1e-006
traingdx, Performance goal met, Figure (11).

**Pass = 10**
traingdx-calcgrad, Epoch 0/300, SSE 0.523948/0.6, Gradient 2.13407/1e-006
traingdx, Performance goal met.

### 2.2.3 Training without Noise Again

Once the network is trained with noise, it makes sense to train it without noise once more to ensure that ideal input vectors are always classified correctly. Therefore, the network is
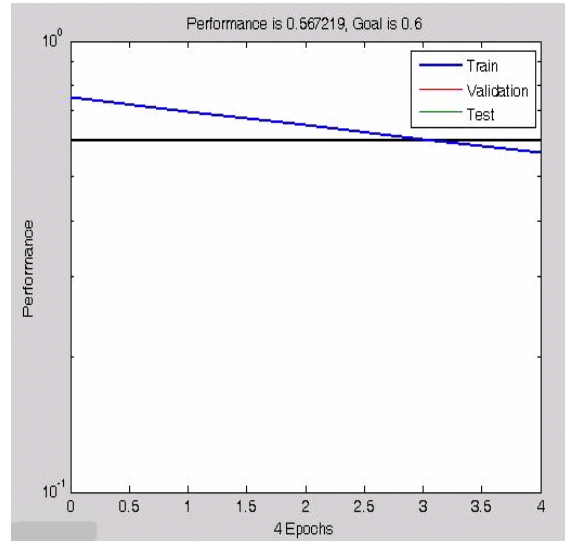


Figure (11): training with noise (Pass9)
again trained with code identical to the previous paragraph 2.3.1.

### 3.0 Results and Discussion

The reliability of the neural network accidents recognition system is measured by testing the network with input vectors with varying quantities of noise. The script file *AccidentsRecogniton* tests the network at various noise levels, and then graphs the percentage of network errors versus noise. Noise with a mean of 0 and a standard deviation from 0 to 0.5 is added to input vectors. At each noise level, 100 presentations of different noisy versions of each accident are made and the network's output is calculated.
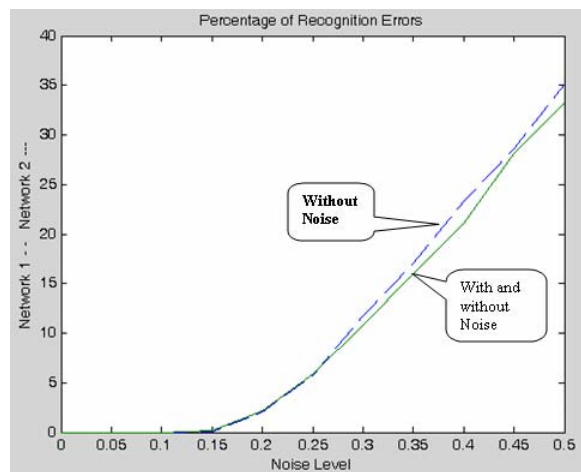


Figure (12): the reliability for the network trained with and without noise

In Figure (12), the *solid* line on the graph shows the reliability for the network trained with and without noise. The reliability of the same network when it was only trained without noise is shown with a *dashed* line. Thus, training the network on noisy input vectors greatly reduces its errors when it has to recognize noisy vectors and its output is approximately equal the training the same network without noisy input vector output.

The network did not make any errors for vectors with noise level of mean 0.00 or 0.14. When noise level of mean is larger than 0.15 was added to the vectors both networks began making errors.

If a higher accuracy is needed, the network can be trained for a longer time, or retrained with more neurons in its hidden layer. Also, the resolution of the input vectors can be increased to a 6-by-10 grid. Finally, the network could be trained on input vectors with greater amounts of noise if greater reliability were needed for higher levels of noise.

To test the system, create an accident with noise and present it to the network. As example, when the accident TR2 (number 3), see figures (1), and add the noise using the *"randn()"* function to generate values from a normal distribution with mean 1 and standard deviation 2, then the output is passed through the competitive transfer function *"compet()"*, that returns a matrix with a 1 in each column where the same column of the input has its maximum value, and 0 elsewhere. Figure (13-A) display the output of the noisy accident TR2, as the noise-free accident TR2 in figures (1), that means the network functioned correctly as expected. Also, when the same test carried out using accident TR4 and TR7, as examples, the same results were given; see figures (13-B), figures (13-C) and all other examples were tested.

```
accidents number=  5
  Columns 1 through 13

     1     1     1     1     1     1     1     0     0     1     0     0

  Columns 14 through 15

     1     1
```

Figures (13-A)

```
accidents number=  3
  Columns 1 through 13

     1     1     0     1     1     0     0     1     1     1     1     1

  Columns 14 through 15

     1     1
```

Figures (13-B)

```
accidents number=  8
  Columns 1 through 13

    1     1     0     0     1     1     1     1     1     1     1     0

  Columns 14 through 15

    0     0
```

Figures (13-C)

Figure (13): Sample Output of the system test when presenting sample of noisy input data to the neural network

The performance of a constructed network can be measured by investigating the network response in more detail, by performing a regression analysis between the network response and the corresponding targets. Where the entire data set are entered the network and a linear regression between the network outputs and the corresponding targets are performed. After the network output and the corresponding targets are passed to the Matlab's function 'postreg', it returns three parameters for the equation:

Output = m Target + b

Figure (12) illustrates the graphical output provided by the function 'postreg'. The network outputs are plotted versus the targets as open circles. The best linear fit is indicated by a dashed line. The perfect fit (output equal to targets) is indicated by the solid line. In this figure, it is difficult to distinguish the best linear fit line from the perfect fit line because the fit is so good.

Output = 0.97 Target + 0.0073

From equation (2), the first parameter is the *slope* m=0.97, and the second one is the *y-intercept* b=0.0073 of the best linear regression relating targets to network outputs. This output, you can see that the numbers are very close to the perfect fit (outputs exactly equal to targets), would be 1, and the *y*-intercept would be 0. The third parameter returned is the *correlation coefficient (R-value)* between the outputs and targets. The correlation coefficient (5) between two variables is a real number (r) which expresses the type and the degree of the relation between the two variables. It is a measure of how well the variation in the output is explained by the targets. If this number is equal to 1, then there is perfect correlation between targets and outputs. In the application, the number is very close to 1 (0.99953), which indicates a good fit.
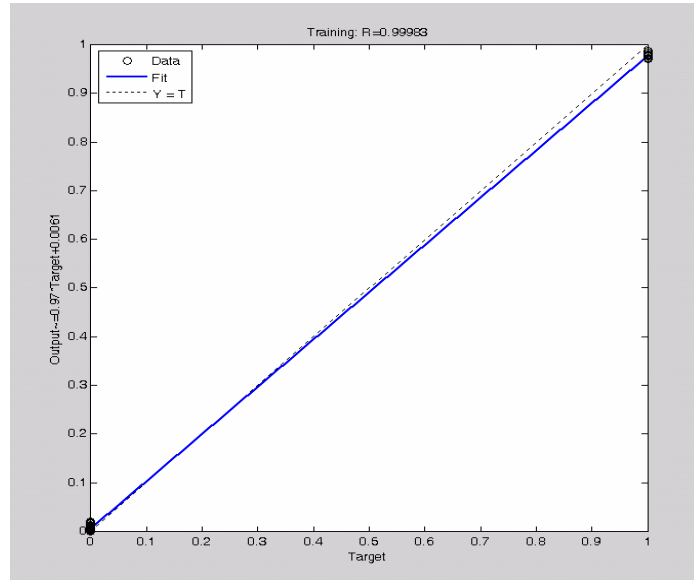
Figure (13): the reliability for the network trained with and without noise

## 4.0 Conclusion

This paper proved the efficient use of an artificial neural network as a promising method for the nuclear reactor accidents patterns recognition. A two layers feedforward neural network with backpropagation training algorithm, which updates the network weights and biases in the direction of the negative of the gradient and trained with an adaptive learning rate combined with momentum training, Matlab's training function 'tringdx', is an efficient ANN to recognize the nuclear reactor accidents patterns.

By testing the network with input vectors with varying values of noise, the network did not make any errors for vectors with noise level of mean 0.00 to 0.14. When noise level of mean is larger than 0.15 was added to the vectors both networks began making errors, and the network still recognize the reactor accidents patterns.

The performance of the constructed network is investigated by performing a regression analysis between the network response and the corresponding targets. The results show the fit is very close to the perfect fit where it is difficult to distinguish the best linear fit line from the perfect fit line, where, the slope m=0.97, the y-intercept b=0.0073, and the *correlation coefficient (R-value)* between the outputs and targets is very close to 1 (0.99953), which indicates the fit is so good.

## 5.0 References

1. S. Sh. Haggag, PhD Thesis, ***"Design and FPGA-Implementation of Multilayer Neural Networks With On-chip Learning"***, Atomic Energy Authority, Egypt 2nd Research Reactor. PhD, Menufia University, 2008.

2. The International Atomic Energy Agency (IAEA) Safety Series, Vienna, "Safety in the Utilization and Modification of Research Reactors", Printed by the IAEA in Austria, December 1994, STI/PUB/961.

3. J. Korbicz, Z. Kowalczuk, J. M. Koscielny, W. Cholewa : "Fault diagnosis: models, artificial intelligence, applications", ISBN 3-540-40767-7, Springer-Verlag Berlin Heidelberg 2004.

4. B. Ch. Hwang, "Fault Detection and Diagnosis of a Nuclear Power Plant Using Artificial Neural Networks", Simon Fraser University, March 1993.

5. The Language Technical Computation (MATLAB), http://www.mathworks.com/, The MathWorks Inc.

6. A. Bartkowiak, "Neural Networks and Pattern Recognition", Institute of Computer Science, University of WrocÃlaw, 2004.

7. M. T. Hagan, H. B. Demuth, "Neural Network Design", PWS Publishing Company, a division of Thomson Learning. United States of America, 1996.

**8.** S. Samarasinghe "Neural Networks for Applied Sciences and Engineering: From Fundamentals to Complex Pattern Recognition", Publisher Auerbach Publishing, ISBN-10: 0-8493-3375-X, Taylor & Francis Group, LLC, 2006.