Compression of Printed English Characters using Back Propagation Neural Network

Sunita[1*], Vaibhav Gupta[1] and Bhupendra Suman[2]
[1] IRDE, DRDO India

sunnidma@yahoo.com, vaibhav.drdo@gmail.com
[2] DEAL, DRDO India

Abstract

In this paper, image data compression algorithm is presented using back propagation neural networks. Back propagation is used to compress printed english characters by training a net to function as an auto associative net (the training input vector and the target output vector are the same) with fewer hidden units than there are in the input or output units. The input and the output data files are formed in +1 and -1 form. The network parameters are adjusted using different learning rates and momentum factors. Mainly, the input pixels are used as target values so that assigned mean square error (MSE) can be obtained, and then the hidden layer output will be the compressed image. The proposed algorithm has been implemented in MATLAB to simulate the algorithm for the english characters A, B, C, D, E. The results obtained, such as compression ratio, mean square error, number of epoch for different learning rates and momentum factors are presented in this paper. Hebbian learning rule and Delta learning rules are used to train the network. Sigmoidal function, binary sigmoidal function and bipolar sigmoidal function are used in feed forward net and back propagation net respectively.

Keyword: Compresssion, Back Propogation Neural Network, Hebbian Learning rule, Sigmoidal function & delta function.

1. Introduction

The main objective of data compression is to decrease the redundancy of the data which helps in increasing the capacity of storage and efficient transmission. Data compression aids in decreasing the size in bytes of a digital data without degrading the quality of the data to a desirable level. The reduction in file size allows more data to be stored in a given amount of disk or memory space. Research activities on neural networks for image compression do exist in many types of networks such as –Multi Layer perceptron (MLP) [1-3], Hopfield [4], Self-Organizing Map (SOM). Artificial neural network models are specified by network topology and learning algorithms [5, 6,7]. Network topology describes the way in which the neurons are interconnected and the way in which they receive input and output. Learning algorithms specify an initial set of weights and indicate how to adapt them during learning in order to improve network performance**.**

In this paper, network has an input layer, a hidden layer and an output layer. The input layer is connected to the hidden layer and the hidden layer is connected to the output layer through interconnection weights. The increase in the number of hidden layers results in computational complexity of the network. As a result, the time required for convergence and to minimize the error can be very high. In this paper, single hidden layer with fewer hidden units than there are in the input or output units are taken for neural network topology to reduce the complexity of the algorithm. The algorithm is simulated for the compression and decompression of the printed alphabetic characters (A – E) which are taken in form of 9 X 7 blocks. In these 9 x 7 blocks, if the pattern of character is recognized then it is +1, and if the pattern is not recognized, it is -1 for input and output files.

2. Proposed Back-Propagation Neural Network

Back-Propagation is a multi-layer forward network using extend gradient-desent based delta-learning rule. Back propagation provides a computationally efficient method for changing the weights in a feed forward network, with differentiable activation functions units, to learn a training set of input-output examples. Being a gradient descent method it minimizes the total squared error of the output computed by the net. The aim of this network is to train the net to achieve a balance between the ability to respond correctly to the input patterns that are used for training and the ability to provide good responses to the input that are similar.

2.1 Architecture

A multilayer feed forward back propagation network having one input layer, one hidden layer and one output layer is considered in this paper [Fig. 1]. The input layer is connected to the hidden layer and the hidden layer is connected to the output layer by means of interconnection weights. The increase in the number of hidden layers results in the computational complexity of the network. As a result, the time taken for convergence and to minimize the error may be very high.

2.2 Training Algorithm

The various parameters used in the training algorithm are as follows:

x : Input training vector

$x = (x_1, \ldots x_i, \ldots x_n)$

t : Output target vector

$t = (t_1, \ldots, t_k, \ldots t_m)$

δ_k = error at output unit y_k

δ_j = error at hidden unit z_j

α= learning rate

V_{oj} = bias on hidden unit j

z_j = hidden unit j

w_{ok} = bias on output unit k

y_k = output unit k

The training algorithm used in the back propagation network involves following four stages:

2.2.1 Initialization of weights

Step 1: Initialize weight to small random values

Step 2 : While stopping condition is false, do step 3 – 10

Step 3: For each training pair do Steps 4 – 9

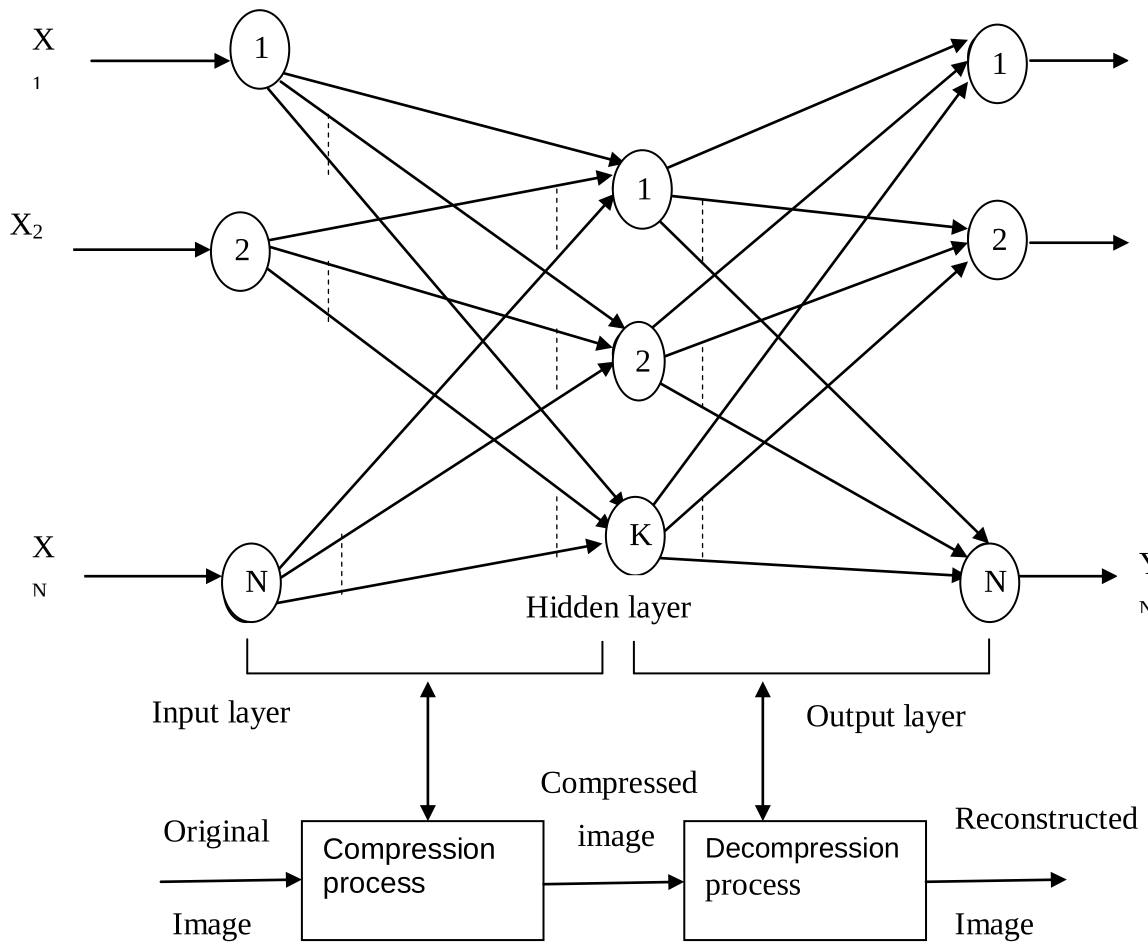


Fig. 1 : BPNN Image Compression System

2.2.2 Feed forward

Step 4: Each input receives the input signal x_i and transmits this signal to all units in the layer above i.e. hidden units

Step 5: Each hidden unit (z_j, j = 1,….p) sums its weighted input signals using Hebbian learning rule and binary sigmoidal function

$$Z_{-inj} = v_{oj} + \sum_{i=1}^{n} x_i\, v_{ij}$$

applying identity function as activation function

$$Z_j = f(z_{-inj})$$

and sends this signal to all units in the layer above i.e. output units.

Step 6: Each output unit (y_k, k=1, ….,m) sums its weighted input signal using Hebbian learning rule and binary sigmoidal function

$$y_{-ink} = w_{ok} + \sum_{j=1}^{p} z_j \quad w_{jk}$$

and applies its identity function as an activation function to calculate the output signals

$$y_k = f(y_{-ink})$$

2.2.3 Back Propagation of errors

Step 7: Each output (y_k, k=1,…,m) receives a target pattern corresponding to an input pattern, error information term is calculated as follows using delta learning rule and bipolar sigmoidal function

$$\delta_k = (t_k - y_k)\ f\ (y_{-ink})$$

Step 8: Each hidden unit (z_j, j = 1, …, n) sums its delta inputs from units in the layer above using Hebbian learning rule and bipolar sigmoidal function

$$\delta_{-inj} = \sum_{k=1}^{m} \delta_j\ w_{jk}$$

The error information term is calculated as

$$\delta_j = \delta_{-inj}\ f\ (z_{-inj})$$

2.2.4 Updation of Weight and Biases

Step 9 : Each output unit (y_k, k = 1, …,m) updates its bias and weights (j =0, …,p)
The weight correction term is given by

$$\Delta\ W_{jk} = \alpha\ \delta_k\ z_j$$

and the bias correction term is given by

$$\Delta\ W_{ok} = \alpha\ \delta_k$$

Therefore, W_{jk} (new) = W_{jk}(old) + $\Delta\ W_{jk}$, W_{ok} (new) = W_{ok}(old) + $\Delta\ W_{ok}$
Each hidden unit (z_j, j =1, …, p) updates its bias and weights (i = 0, …n)

The weight correction term

$$\Delta\ V_{ij} = \alpha\ \delta_j\ x_i$$

The bias correction term

$$\Delta\ V_{oj} = \alpha\ \delta_j$$

Therefore, V_{ij} (new) = V_{ij} (old) + $\Delta\ V_{ij}$, V_{oj} (new) = V_{oj} (old) + $\Delta\ V_{oj}$
Step 10: Test the stopping condition.

The stopping condition may be the minimization of the errors, numbers of epochs etc.

3 Selection of Parameters

For the efficient operation of the back propagation network, it is necessary for the appropriate selection of the parameters used for training. There are a number of different parameters that must be decided upon when designing a neural network. Among these parameters are the number of layers, the number of neurons per layer, the number of training iterations, etc. Some of the more important parameters in terms of training and network capacity are the number of hidden neurons, the learning rate and the momentum factor. The initial value assignments are discussed in this section. The details of these parameters are as follows :

3.1 Number of Neurons in the Hidden Layer

Hidden neurons are the neurons that are neither in the input layer nor the output layer. These neurons are essentially hidden from view, and their number and organization can typically be treated as a black box to people who are interfacing with the system. Using additional layers of hidden neurons enables greater processing power and system flexibility. This additional flexibility comes at the cost of additional complexity in the training algorithm. Having too many hidden neurons is analogous to a system of equations with more equations than there are free variables: the system is over specified, and is incapable of generalization. Having too few hidden neurons, conversely, can prevent the system from properly fitting the input data, and reduces the robustness of the system. In this BPNN algorithm, we are taking one hidden layer. For simulation result, different numbers of hidden neurons in one hidden layer are taken as 24, 14, and 10.

3.2. Learning Rate

Learning rate is the training parameter that controls the size of weight and bias changes during learning. In this algorithm, we have taken two different values of learning rate parameter as 0.3 and 0.5.

3.3 Momentum factor

Momentum factor simply adds a fraction m of the previous weight update to the current one. The momentum parameter is used to prevent the system from converging to a local minimum or saddle point. A high momentum parameter can also help to increase the speed of convergence of the system. However, setting the momentum parameter too high can create a risk of overshooting the minimum, which can cause the system to become unstable. A momentum coefficient that is too low cannot reliably avoid local minima, and can also slow down the training of the system. 0.2 and 0.4 are taken as momentum factor in the designed algorithm.

3.4 Training type

0 = train by epoch

1 = train by minimum error

3.5 Epoch

Determines when training will stop once the number of iterations exceeds epochs. When training by minimum error, this represents the maximum number of iterations. We have taken 100 and 60 epoch for minimum error 0.008.

3.6 Minimum Error

Minimum mean square error of the epoch is called minimum error. It can be defined as the square root of the sum of squared differences between the network targets and actual outputs divided by number of patterns (only for training by minimum error). In this algorithm we have taken minimum squared error (MSE) as 0.008.

4 Methodology for Image Data Compression using Proposed BPNN

The main idea of neural network applications for the problem of image compression is to construct such a network in which the input and output layer with the same number of neurons will be the connection to the middle (hidden) layer with smaller number of neurons, which is the precondition for compression realization. The relation input/hidden layer size is the compression rate. The goal of such compression is to reconstruct its input hence the output of the network that solves this class of problems is equal to its input. Network training starts with initialization of its weights and requires a set of examples with appropriate input-output pairs. Weights in the network are iteratively modified during the training time to minimize the performance function of the given network, which is taken as MSE (Mean Square Error between the calculated and expected network output). The procedure is repeating till deviation between calculated and expected output is 0.008.

4.1 Steps of BPNN algorithm: A simulation program to implement back propagation was built which include the following steps:

1. Take 63 input neurons of block size 9 x 7 to form a character for training vector pair (input and the corresponding output) in training set files. For simulation 05 training vector pairs are taken.
2. Initialization of network weights to small random values, learning rate (alpha) and threshold error. Set iterations to zero.
3. Open training set file.
4. Total_error = zero; iterations → iterations+1.
5. Get one vector and feed it to input layer.
6. Get the target output of that vector.
7. Calculate the outputs of hidden layer units.
8. Calculate the outputs of output layer units.
9. Calculate error (desired output - actual output) and calculate total_error → total_error + error.
10. Calculate delta sigma of output layer units and adjust weights between output and hidden layer.
11. Calculate delta sigma of hidden layer units and adjust weights between hidden and input layer.
12. While there are more vectors, go to Step 5.
13. If threshold error >= total error then stop, otherwise go to Step 4.

For simulation, 5 alphabetical characters (A – E) are chosen. Neural network with 24, 14 and finally with 10 neurons in hidden layers are trained to achieve compression rates 3, 5, and 6 respectively.

5 Simulation Result

For simulation result, 5 alphabetical characters (A – E) are chosen. Block size of one character is 9 x 7 i.e. in input layer there are 63 neurons. Input patterns are shown in table 1. Network has been trained for 24, 14, and 10 neurons in hidden layer. Learning rate (0.3 , 0.5) and momentum factors (0.2, 0.4) are also adjusted to achieve the desired (0.008) minimum squared error. Most of the characters from “A” to “E” get their shape almost similar to the original shape after decompression.

Table 2, 3 and 4 shows the minimum squared error (MSE) achieved by characters “A” – “E” in compression for different combinations of learning rates, momentum factor, epoch and compression ratio.

Graphical simulation result for compression ratio 3, 5 and 6 have been taken. Section 5.1 shows the graphical simulation result for compression ratio 3 only to avoid repetition of the results. Algorithm is implemented in Matlab.

Table 1 Representation of Input Data

A	[-1-1 1 1 -1-1-1 -1-1-1 1-1-1-1 -1-1-1 1-1-1-1 -1-1 1-1 1-1-1 -1-1 1-1 1-1-1 -1 1 1 1 1 1-1 -1 1-1-1-1 1-1 -1 1-1-1-1 1-1 1 1 1-1 1 1 1]
B	[1 1 1 1 1 1-1 -1 1-1-1-1-1 1 -1 1-1-1-1-1 1 -1 1-1-1-1-1 1 -1 1 1 1 1 1-1 -1 1-1-1-1-1 1 -1 1-1-1-1-1 1 -1 1-1-1-1-1 1 1 1 1 1 1 1-1]
C	[-1-1 1 1 1 1-1 -1 1-1-1-1-1 1 1-1-1-1-1-1-1 1-1-1-1-1-1-1 1-1-1-1-1-1-1 1-1-1-1-1-1-1 1-1-1-1-1-1-1 -1 1-1-1-1-1 1 -1-1 1 1 1 1 1]
D	[1 1 1 1 1-1-1 -1 1-1-1-1 1-1 -1 1-1-1-1-1 1 -1 1-1-1-1-1 1 -1 1-1-1-1-1 1 -1 1-1-1-1-1 1 -1 1-1-1-1-1 1 -1 1-1-1-1 1-1 1 1 1 1 1 -1-1]
E	[1 1 1 1 1 1 1 -1 1-1-1-1-1 1 -1 1-1-1-1-1-1 -1 1-1 1-1-1-1 -1 1 1 1-1-1-1 -1 1-1 1-1-1-1 -1 1-1-1-1-1-1 -1 1-1-1-1-1 1 1 1 1 1 1 1 1]

Table 2 Data compression results in terms of Minimum Squared Error (MSE) for characters A – E. Compression ratio = 3.

Character	Learning Rate = 0.3 Momentum Factor = 0.2 Epoch = 60 **MSE**	Learning Rate = 0.5 Momentum Factor = 0.4 Epoch = 60 **MSE**	Learning Rate = 0.3 Momentum Factor = 0.2 Epoch = 100 **MSE**	Learning Rate = 0.5 Momentum Factor = 0.4 Epoch = 100 **MSE**
A	0.085339	0.006627	0.046953	0.005225
B	0.015665	0.076973	0.072312	0.007473
C	0.015560	0.081357	0.057334	0.008974
D	0.019492	0.103078	0.058548	0.006590
E	0.014536	0.083399	0.057228	0.006352

Table 3 Data compression result in terms of Minimum Squared Error (MSE) for characters A – E. Compression ratio = 5

Character	Learning Rate = 0.3 Momentum Factor = 0.2 Epoch = 60 **MSE**	Learning Rate = 0.5 Momentum Factor =0.4 Epoch = 60 **MSE**	Learning Rate = 0.3 Momentum Factor =0.2 Epoch = 100 **MSE**	Learning Rate = 0.5 Momentum Factor = 0.4 Epoch = 100 **MSE**
A	0.163688	0.034051	0.079771	0.022048
B	0.140148	0.057788	0.074416	0.020125
C	0.184303	0.029748	0.089696	0.028202
D	0.173084	0.030384	0.109465	0.026873
E	0.121639	0.047957	0.077747	0.015614

Table 4 Data compression result in terms of Minimum Squared Error (MSE) for characters A – E. Compression ratio = 6

Character	Learning Rate = 0.3 Momentum Factor = 0.2 Epoch = 60 **MSE**	Learning Rate = 0.5 Momentum Factor = 0.4 Epoch = 60 **MSE**	Learning Rate = 0.3 Momentum Factor = 0.2 Epoch = 100 **MSE**	Learning Rate = 0.5 Momentum Factor = 0.4 Epoch = 100 **MSE**
A	0.187018	0.065062	0.100105	0.039388
B	0.275004	0.042760	0.133900	0.031317
C	0.216361	0.077609	0.100406	0.042369
D	0.188603	0.047558	0.123496	0.071309
E	0.189438	0.057925	0.098847	0.031682

5.1 Graphical Simulation Result for letter "A"- "E". Compression ratio = 3

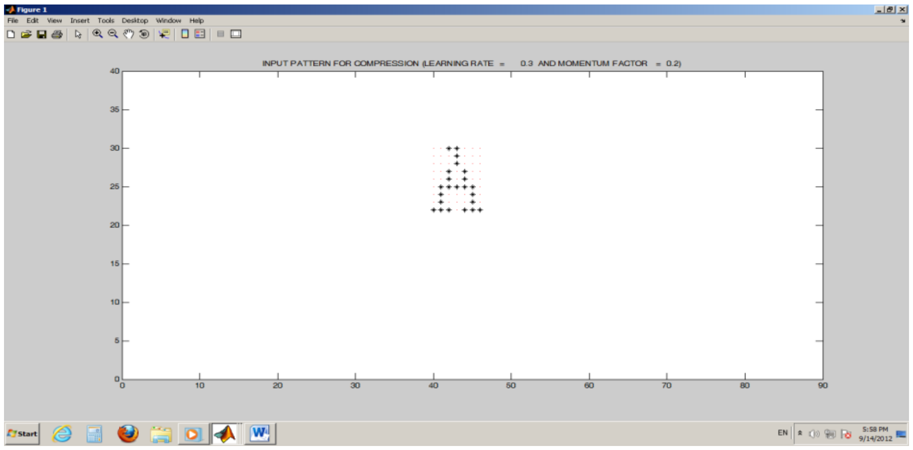


Figure 1
File Edit View Insert Tools Desktop Window Help
INPUT PATTERN FOR COMPRESSION (LEARNING RATE = 0.3 AND MOMENTUM FACTOR = 0.2)
Start
EN

Input Pattern for Character 'A"

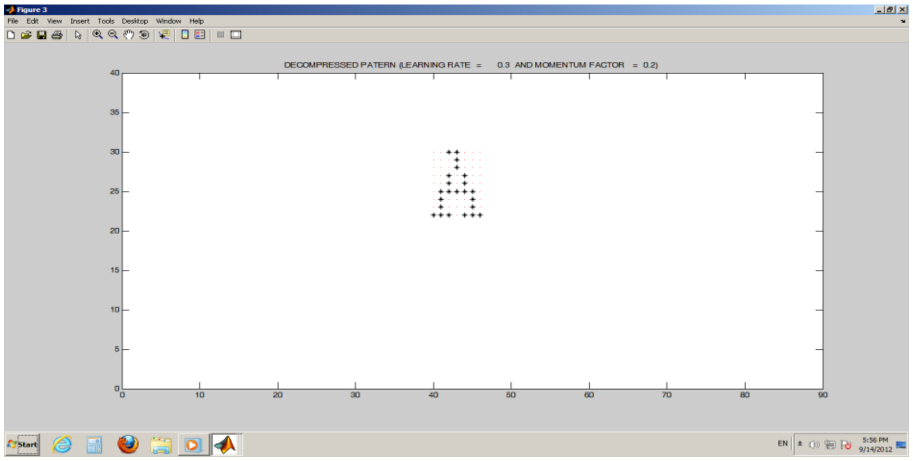
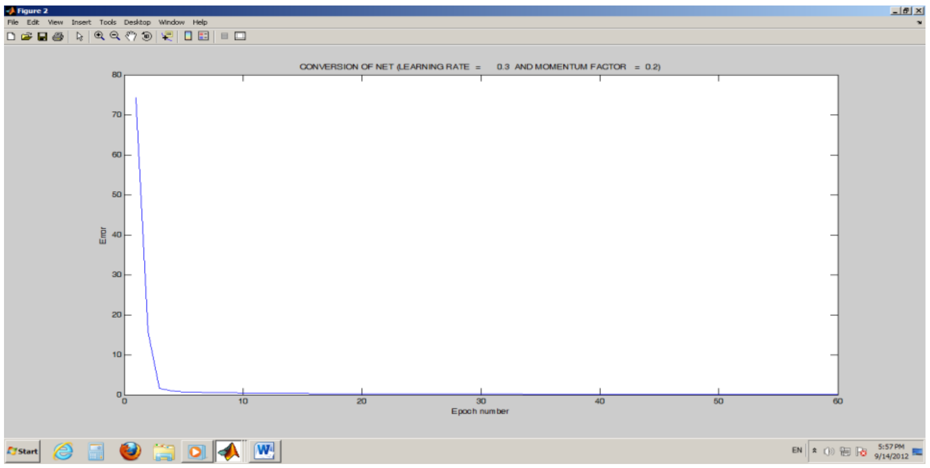
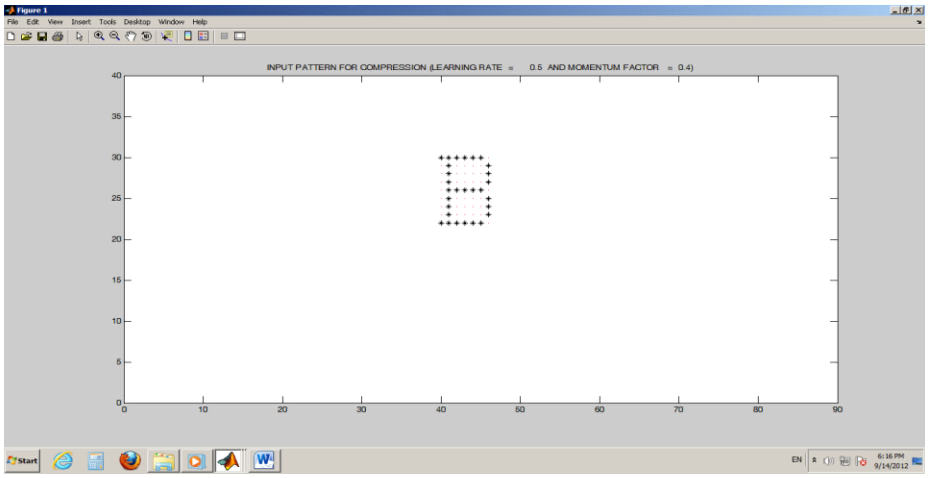


Figure 3
File Edit View Insert Tools Desktop Window Help
DECOMPRESSED PATERN (LEARNING RATE = 0.3 AND MOMENTUM FACTOR = 0.2)
Start
EN

Decompressed Pattern (Learning Rate = 0.3 and Momentum Factor = 0.2)



Conversion of NET (Learning Rate = 0.3 and Momentum Factor = 0.2)


INPUT PATTERN FOR COMPRESSION (LEARNING RATE = 0.5 AND MOMENTUM FACTOR = 0.4)

Conversion of NET (Learning Rate = 0.3 and Momentum Factor = 0.2)

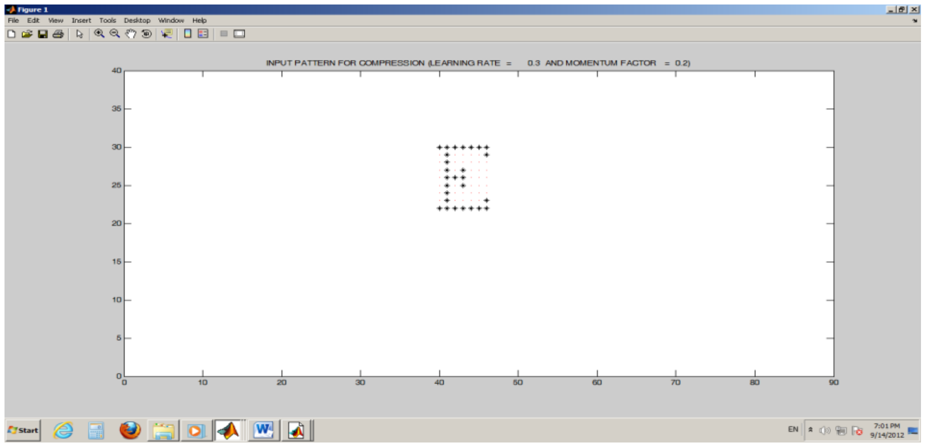


Figure 1
File Edit View Insert Tools Desktop Window Help
INPUT PATTERN FOR COMPRESSION (LEARNING RATE = 0.3 AND MOMENTUM FACTOR = 0.2)
0 5 10 15 20 25 30 35 40
0 10 20 30 40 50 60 70 80 90
Start
EN
7:01 PM
9/14/2012

Input Pattern for Character ‘E”

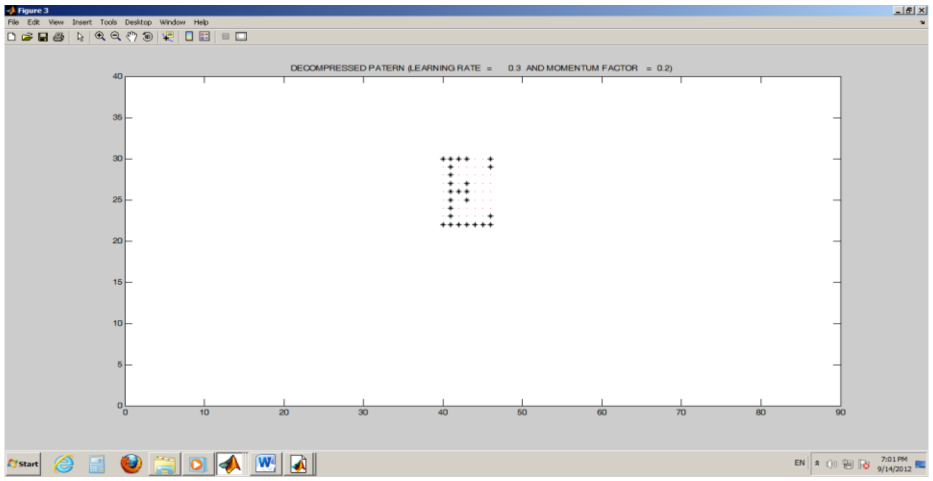
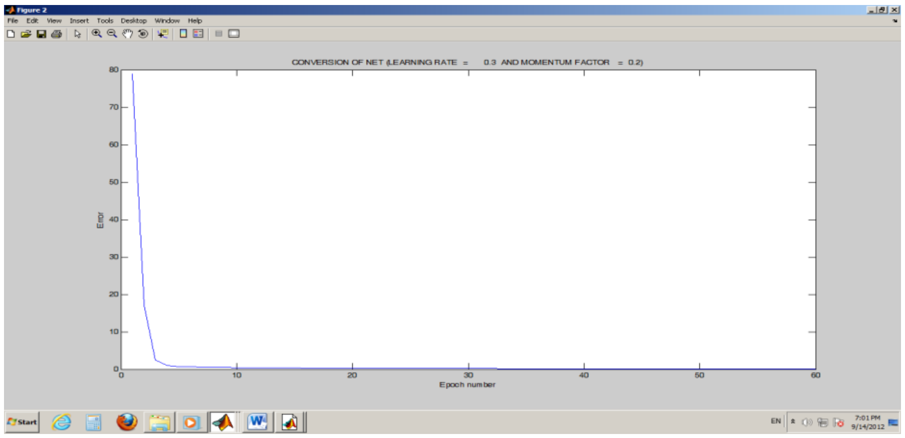


Figure 3
File Edit View Insert Tools Desktop Window Help
DECOMPRESSED PATERN (LEARNING RATE = 0.3 AND MOMENTUM FACTOR = 0.2)
0 5 10 15 20 25 30 35 40
0 10 20 30 40 50 60 70 80 90
Start
EN
7:01 PM
9/14/2012

Decompressed Pattern (Learning Rate = 0.3 and Momentum Factor = 0.2)



Conversion of NET (Learning Rate = 0.3 and Momentum Factor = 0.2)

6 Conclusion

The performance of the designed BPNN image data compression algorithm to the desired minimum squared error is achieved by modifying the network itself by using different values of learning rate parameters, momentum factors, number of neurons in hidden layer and epoch. Combination of Hebbian learning rule and Delta learning rule support the algorithm to converge to the MSE (0.008) with epoch (100).

7 References

1. Aree Ali Mohammed, Miran Taha Abdullah. “Compressed Medical Image Transfer in Frequency Domain.” International Journal of Image Processing (IJIP) 5 (2011): 371-379.
2. C.H. Chen, Y.Yao, D.Page, B.Abidi, A.Koschan, M.Abid. “Comparison of Image Compression Methods Using Objective Measures towards Machine Recognition.” IJIAP International Journal of Information Analysis and Processing 1 (2008): 63-74.
3. S.S. Panda, M.S.R.S Prasad, MNM Prasad, Ch. SKVR Naidu. “Image Compression using Back Propagation Neural Network.” International Journal of Engineering Science & Advanced Technology. 2 (2012): 74-78.
4. Suprava Patnaik, R. N. Pal. "Image compression using Auto associative neural network and embedded zero-tree coding" Third IEEE processing workshop advances in wireless communication, Taiwan. (2001).
5. Mukesh Mittal, Ruchika Lamba, “Image Compression Using Vector Quantization Algorithms”. International Journal of Advanced Research in Computer Science and Software Engineering. Volume 3, Issue 6, June 2013

Books

6. Simon Haykin, Neural Networks – A comprehensive Foundation. New Delhi : Pearson Prentice Hall, 2004.
7. SN Sivanandam, S Sumathi, SN Deepa Introduction to Neural Network using Matlab 6.0. New Delhi: Tata McGraw-Hill Publishing Company Limited, 2006.