First Joint Congress on Fuzzy and Intelligent Systems
Ferdowsi University of Mashhad, Iran
29-31 Aug 2007

Finding the Inversion of a Square Matrix and Pseudo-inverse of a Non-square Matrix by Hebbian Learning Rule

Zeinab Ghassabi, Ali Khaki-Sedigh
Tehran Azad University, Science and Research Branch, KNTU
rose_z60gh@yahoo.com, sedigh@kntu.ac.ir

Abstract: In this paper, we discuss a neural network based on hebbian learning rule for finding the inverse of a matrix. First we described finding the inverse of a matrix by mentioned neural network. Finally, experimental results for square and non-square matrices are presented to show the effectiveness of the approach. Proposed method is also scalable for finding the inversion of large-scale matrices.

Keywords: Hebbian Learning rule, neural networks, solving simultaneous linear equations

1 Introduction

Finding the inversion of a matrix is considered to be one of the basic problems widely encountered in science and engineering since it is frequently used in many applications (e.g. robotics, signal processing, Wiener and Kalman filtering). In many applications an on-line (e.g. real time) inversion of a matrix is desired.

In monitoring and control of dynamic systems, there is often a need for finding a real-time matrix inversion of a large-scale matrix. If such a large-scale problem needs to be solved in real time, existing numerical methods and sequential techniques may not scale well.

There are a variety of methods for matrix inversion, usually classified as direct and iterative. Direct methods are distinguished by the Fact that they find the correct solution in a finite number of operations. Iterative methods are characterized by an initial estimate of the solution, and subsequent update of the estimate based on the previous estimate and some error measure. Iterative algorithms are sometimes preferred, especially in problems of medium/large size [7].

Finding the inversion of a matrix with constant and time varying coefficients has been studied in the field of neural networks [1, 2, 3] with varying results in a highly parallel fashion. Recurrent Neural Network (RNN) and neural networks with perceptron learning rule were studied to find matrix inversion. RNN (i.e. continuous approach) recovers its inputs to find matrix inversion while in the neural networks with perceptron learning rule (i.e. discontinuous approach), the matrix weight is updated until all the outputs of neurons are same as desire ones.

In this paper, a one-layer neural network that updates its weights by hebbian learning rule was applied for inverting of square and nonsquare matrices.

Hebbian learning rule is one of the first unsupervising learning rules of neural networks and it is the base of all rules used for associative neural networks. This rule first proposed by Donald Heb in 1949 [5].

The rest of the paper is organized as follows: Section 2 presents problem formulation, while the neural network model and hebbian learning rule for inverting of matrices is presented in section 3 with experimental results discussed in section 4.

2 Problem Formulation to Find matrix Inversion

2.1 A is a square Matrix

In this case, finding the inversion of matrix A is the same as solving simultaneous equations as follows [2]:

$$AB = I \qquad (1)$$

that

$$A = \begin{bmatrix} a_{11}, a_{12}, \dots, a_{1n} \\ a_{21}, a_{22}, \dots, a_{2n} \\ \vdots \\ a_{n1}, a_{n2}, \dots, a_{nn} \end{bmatrix}, \quad B = \begin{bmatrix} b_{11}, \dots, b_{1n} \\ b_{21}, \dots, b_{2n} \\ \vdots \\ b_{n1}, \dots, b_{nn} \end{bmatrix}, \quad I = \begin{bmatrix} 1,0,\dots,0 \\ 0,1,\dots,0 \\ \vdots \\ 0,\dots,0,1 \end{bmatrix}$$

$I_{n\times n}$, $B_{n\times n}$ are identity matrix and inversion of matrix A respectively ($AA^{-1} = I$).

2.2 A is a nonsquare Matrix

When A is a nonsquare matrix, the simultaneous equations for finding the inversion of matrix A (A is a $m \times n$ matrix that $m > n$) is as follows:

$$BA = I \qquad (2)$$

that

$$A = \begin{bmatrix} a_{11}, a_{12}, \dots, a_{1n} \\ a_{21}, a_{22}, \dots, a_{2n} \\ \vdots \\ a_{m1}, a_{m2}, \dots, a_{mn} \end{bmatrix}, \quad b = \begin{bmatrix} b_{11}, \dots, b_{1n} \\ b_{21}, \dots, b_{2n} \\ \vdots \\ b_{n1}, \dots, b_{nm} \end{bmatrix}, \quad I = \begin{bmatrix} 1,0,\dots,0 \\ 0,1,\dots,0 \\ \vdots \\ 0,\dots,0,1 \end{bmatrix}$$

In this case, the pseudoinverse of A is a $n \times m$ matrix as follows:

$$pinv = (A^T A)^{-1} A^T \qquad (3)$$

that

$$pinv_{n\times m} \times A_{m\times n} = I_{n\times n} \qquad (4)$$

It is required to construct the algorithm finding for any matrix A any value of the matrix B satisfying all the linear equations in Equations (1) and (2) simultaneously.

We wish to design a neural network to solve the systems of above linear equations. In the following section, we will present the structure of neural network and its learning rule to solve Equations (1) and (2).

3 The neural network model base hebbian learning rule for finding the inversion of square and nonsquare matrices.

Equation (1) can be rewritten as follows:

$$AW = I \qquad (5)$$

and Equation (2) can be rewritten as follows:

$$WA = I \qquad (6)$$

Matrix W is the solution of the systems of linear equations in Equations (5) and (6).

The neural network model for solving Equations (5) and (6) is a one-Layer neural network, with the n input variables as $a_{i1}, a_{i2}, \dots, a_{in}$ that $i = 1, \dots, n \times n$ ($n \times n$ is the number of patterns in neural network), matrix I of bias inputs (hebbian rule is unsupervising learning rule) or an additional inputs which have weight -1 and the weight matrix as $w_{1j}, w_{2j}, \dots, w_{nj}$ (A is a square matrix).

If A is a nonsquare matrix, the neural network model has $n \times n$ patterns for inputs and the dimension of each pattern is m.

The neural network model for finding the inversion of a matrix by hebbian learning rule, is showed in Figure 1 (A is a square matrix).

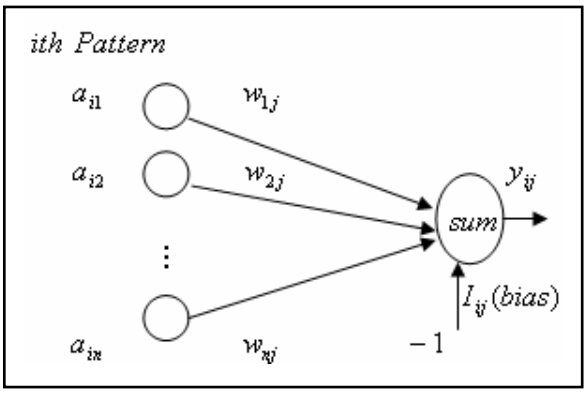

Figure 1: Neural network model for inversion of a matrix (A is a square matrix)

The activation function in neural network is linear .The output neuron (linear unit) in Figure 1 is weighted sum of input variables as follows:

$$y_{ij} = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} w_{ji} - I_{ij} \qquad (7)$$

and weights will be updated as follows:

$$\Delta w_{ij} = -\varepsilon a_{ij} y_{ij} \quad where \quad \varepsilon > 0 \qquad (8)$$

that $\varepsilon > 0$ is learning rate and a small constant (when A is a square matrix).

If A is a nonsquare matrix, The output neuron in Figure 1 is weighted sum of input variables as follows:

$$y_{ij} = \sum_{j=1}^{m} \sum_{i=1}^{n} w_{ji} a_{ij} - I_{ij} \qquad (9)$$

and weights will be updated as Equation (8).

In the network weight vector, $w_{1j}, w_{2j}, \dots, w_{nj}$, is going to close to vector $b_{1j}, b_{2j}, \dots, b_{nj}$ ($\min \|B - W\|$) by iterating procedure of the learning of neural network based on hebbian learning rule. Therefore, matrix W is the solution of Equations (1) and (2).

4 Experimental Results

In the simulation, $\varepsilon = 0.01$ and the initial weights were chosen between -0.05 and 0.05 randomly.
One of the key concepts in numerical analysis is the conditioning of problem. For some matrices by adding small perturbation E and finding the inverse of the perturbed system $((A+E)X = I$, we noticed that how much the small change in the data is magnified in the solution. One way to measure the magnification factor is by means of the quantity $\|A\|\|A^{-1}\|$ called the condition number of with respect to inversion.
The proposed approach examine for matrices with different value of condition numbers. For 20 matrices with condition numbers smaller than 5, the number of iterations are smaller than 50 (Lrate=0.01). The numbers of iterations in comparison with RNN approaches in [7] for the same condition number are less.

4.1 A is a square Matrix

Example 1: the inversion of matrix $A_{3\times3}$ (Condition number is 31.4380.) is computed as follows:

$$\begin{bmatrix} 1 & 2 & -1 \\ 1 & 1 & -1 \\ 1 & 6 & 7 \end{bmatrix}_{3\times3} \begin{bmatrix} w_{11}, w_{12}, w_{13} \\ w_{21}, w_{22}, w_{23} \\ w_{31}, w_{32}, w_{33} \end{bmatrix}_{3\times3} = \begin{bmatrix} 1,0,0 \\ 0,1,0 \\ 0,0,1 \end{bmatrix}_{3\times3}$$

We have $3\times3=9$ patterns as follows:

$input_1 = 1,2,-1 \quad desire_1 = 1$

$input_2 = 1,2,1 \quad desire_2 = 0$

$input_3 = 1,2,1 \quad desire_3 = 0$

$input_4 = 1,1,-1 \quad desire_4 = 0$

$input_5 = 1,1,-1 \quad desire_5 = 1$

$input_6 = 1,1,-1 \quad desire_6 = 0$

$input_7 = 1,6,7 \quad desire_7 = 0$

$input_8 = 1,6,7 \quad desire_8 = 0$

$input_9 = 1,6,7 \quad desire_9 = 1$

Program code in MATLAB7 is as follows:

```
program matrix_inversion
  W=0.05*rand(3,3); %weight matrix
  Lrate=0.01; %learning rate
  iter=12900; %iterations
  for t=1:iter
    for i=1:3
      for j=1:3
        y=A(i,:)*W(:,j)-I(i,j);
        W(:,j)=W(:,j)-(Lrate*A(i,:)*y)'
      end;  % of for loop j
    end;  % of for loop i
  end; % of for loop t
```

Computations for first pattern are as follows:

$$y = A(:,1)\times W(1,:) - I(1,1) = 1\times W_{11} + 2\times W_{12} + 1\times W_{13} - 1$$

$$W(:,1) = W(:,1) - \varepsilon \times A(1,:)^T \times y$$

After 12500 iterations the exact solution according to the Equations (7) and (8) is as follows:

$$W = \begin{bmatrix} -1.6250 & 2.5000 & 0.1250 \\ 1.0000 & -1.0000 & 0.0000 \\ -0.6250 & 0.5000 & 0.1250 \end{bmatrix}$$

The initial weights were as follows:

$$W = \begin{bmatrix} 0.0265 & 0.0190 & 0.0231 \\ 0.0320 & 0.0392 & 0.0284 \\ 0.0105 & 0.0340 & 0.0397 \end{bmatrix}$$

Figure 2 illustrates weight matrix until to reach to the solutions (matrix inversion).

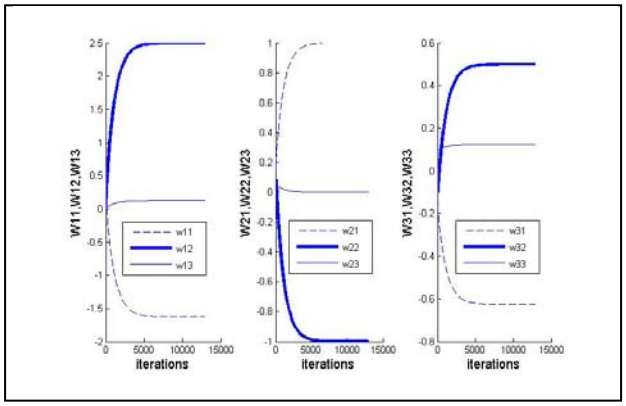

Figure 2: Weights until to reach to the solutions for matrix $A_{3\times3}$

Example 2: the inversion of matrix $A_{6\times6}$ (Condition number is 46.3729 and the coefficients of matrix are real) is computed as follows:

$$A = \begin{bmatrix} 1.2 & 5.6 & 0.0 & 4.2 & 3.4 & 7.2 \\ 2.6 & 0.0 & 2.4 & 0.0 & 3.7 & 8.2 \\ 9.1 & 2.7 & 4.3 & 0.0 & 4.3 & 0.0 \\ 0.0 & 3.6 & 5.2 & 4.8 & 9.6 & 1.3 \\ 5.6 & 0.0 & 3.5 & 0.0 & 5.2 & 6.8 \\ 1.4 & 6.3 & 0.0 & 3.8 & 6.8 & 4.4 \end{bmatrix}_{6\times6}$$

We have $6\times6=36$ patterns and the exact inversion by MATLAB7 and proposed approach is as follows:

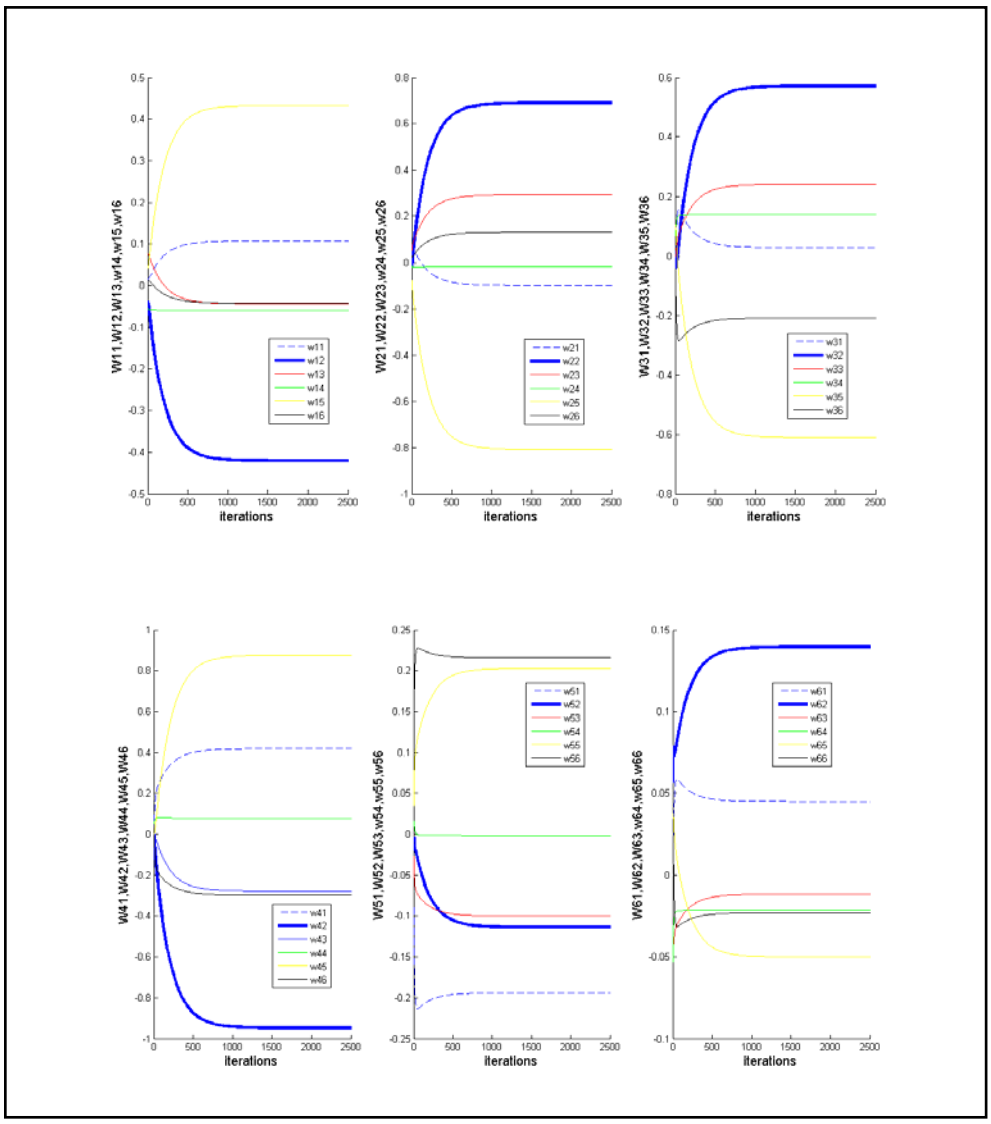

Figure 3: Weights until to obtain the inversion of a large scale matrix

$$W = \begin{bmatrix} 0.1075 & -0.4208 & -0.0437 & -0.0607 & 0.4325 & -0.0421 \\ -0.0989 & 0.6897 & 0.2933 & -0.0168 & -0.8083 & 0.1305 \\ 0.0282 & 0.5712 & 0.2409 & 0.1404 & -0.6106 & -0.2086 \\ 0.4188 & -0.9465 & -0.2778 & 0.0773 & 0.8755 & -0.2973 \\ -0.1937 & -0.1137 & -0.1000 & -0.0014 & 0.2028 & 0.2158 \\ 0.0450 & 0.1395 & -0.0115 & -0.0212 & -0.0499 & -0.0230 \end{bmatrix}$$

The proposed approach is scalable for finding the inversion of large-scale matrices.

Example 3: consider an upper triangular matrix $A_{20\times 20}$ (condition number is 26.0312) of ones (zero below the diagonal) given by $A = triu\,(ones\,(20))$ in MATLAB7 (Condition number is 26.0312). The results are given in Table 1.

Table 1: Results of Matrix $A_{20\times 20}$

Number of Iterations	Learning Rate of Hebbian Learning Rule
<3500	0.01
<900	0.1

Some results are given in Table 2 about the relation of dimensions of matrices with different condition numbers to the number of iterations of hebbian learning rule for inverting of matrices.

According to Table 2, if the value of the condition number reduced, the number of iterations will be

smaller (dimensions and learning rate were not changed) and if the value of the Learning rate reduced, the number of iterations will be greater (dimensions and condition number were not changed). The affect of condition number in changing the number of iterations are more than the affect of learning rate and dimension of matrices.

Table 2: Relation of Some Matrices with Different Condition Numbers to the Number of Iterations

Dimension of a Matrix (condition number)	Number of Iterations (Learning Rate)
2×2(1.2937)	30(0.01)
2×2(1.2937)	350(0.001)
3×3(31.4380)	12500(0.01)
3×3(9.35)	862(0.01)
3×3(1.8006)	31(0.01)
20×20(26.0312)	<3500(0.01)
20×20(26.0312)	<900(0.1)

4.2 A is a nonsquare Matrix

The inversion of matrix $A_{3\times 2}$ (Condition number is 4.2159.) is computed as follows:

$$\begin{bmatrix} w_{11}, w_{12}, w_{13} \\ w_{21}, w_{22}, w_{23} \end{bmatrix}_{2\times 3} \begin{bmatrix} 1 & 8 \\ 6 & 8 \\ 3 & 5 \end{bmatrix}_{3\times 2} = \begin{bmatrix} 1,0 \\ 0,1 \end{bmatrix}_{2\times 2}$$

We have $2 \times 2 = 4$ patterns and computations for first pattern are as follows:

$$y = W(1,:) \times A(:,1) - I(1,1) = W_{11} \times 1 + W_{12} \times 6 + W_{13} \times 3 - 1$$

$$W(1,:) = W(1,:) - \varepsilon \times A(:,1)^T \times y$$

After 63 iterations the solution according to Equations (8) and (9) is as follows:

$$W = \begin{bmatrix} -0.2109 & 0.1655 & 0.0727 \\ 0.1487 & -0.0293 & 0.0090 \end{bmatrix}$$

that

$$\begin{bmatrix} -0.2109 & 0.1655 & 0.0727 \\ 0.1487 & -0.0293 & 0.0090 \end{bmatrix}_{2\times 3} \begin{bmatrix} 1 & 8 \\ 6 & 8 \\ 3 & 5 \end{bmatrix}_{3\times 2} = \begin{bmatrix} 1,0 \\ 0,1 \end{bmatrix}_{2\times 2}$$

The initial weights were as follows:

$$W = \begin{bmatrix} 0.0171 & 0.0171 & 0.0364 \\ 0.0145 & -0.0267 & 0.0155 \end{bmatrix}$$

W by using $pin(A)$ command in MATLAB7 is as follows:

$$W = \begin{bmatrix} -0.2078 & 0.1753 & 0.0521 \\ 0.1487 & -0.0290 & 0.0085 \end{bmatrix}$$

Figure 4 illustrates weight matrix until to reach to the solutions (the pseudoinverse of a matrix).

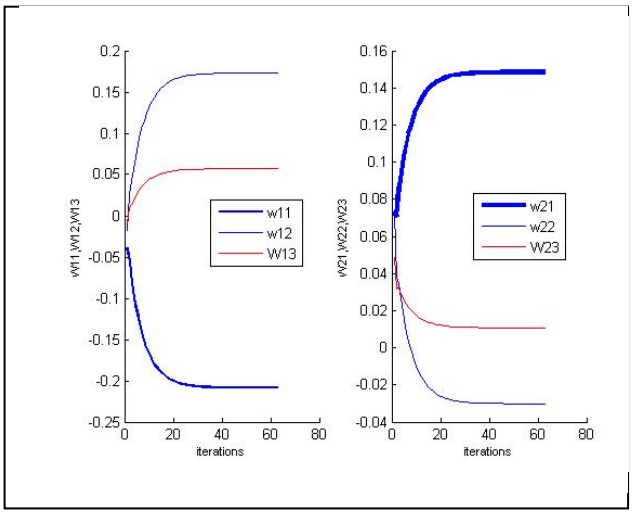

Figure 4: Weights until to reach to the pseudoinverse of matrix $A_{3\times 2}$

5 Conclusions

Using hebbian learning rule for finding the inversion of a matrix is shown by experiments to be very effective and feasible method especially for large-scale problems (implementing by discontinuous approaches), due to smaller storage requirements and efficiency of computational time. The fast training speed (especially for matrices with condition numbers smaller than 5 in comparison with proposed RNNs in [7]) and high accuracy are the most important advantages for this method. The speed of convergence of neural network also depends on appropriate selections of learning rate [8].

References

[1] A. Cichocki and R. Unbenhaun, "Neural Networks for Solving Systems of Linear Equations And Related Problems," *IEEE Trans. Circuits and Systems-I*, Vol. 39, pp. 124-138, Feb. 1992.

[2] Z. Ghassabi, “Solving systems of linear equations by artificial intelligence approaches,” M.S. thesis, Dept. Computer. Eng., Azad Univ., Science and Research Branch, Tehran, Iran, 2006.
[3] D. S. Haung,“ One-Layer Linear Perceptron for the Inversion of Nonsingular Matrix,” *Proc. Int. Conf.* ROVPIA'96, Ipoh, Malaysia, pp. 639–643, 1996.
[4] D. S. Huang and Z. Chi, “Solving Linear Simultaneous Equations by Constraining Learning Neural Networks,” *IEEE Int. Conf. on Neural Networks*, pp. 775-779, 1995.
[5] M. B. Menhaj, *Computational intelligence.* Tehran, IRAN: Amirkabir University, 2002.
[6] J. Wang, “Electronic realization of recurrent neural network for solving simultaneous linear equations,” *IEEE Electronic Letters*, Vol. 28, pp. 493-495, 1999.
[7] Wang J., “A Recurrent Neural Network for real-time matrix inversion,” *Appl. Morh. Compur.* Vol. 26, pp. 89-100, 1993.
[8] K. Zhang, G. Ganis, and M. I. Sereno,“ Anti-Hebbian synapses as a linear equation solver, ” *Proc. IEEE Int. Conf. on Neural Networks*, pp. 387-389, 1997.