ISSN: 0975-766X
CODEN: IJPTFI
Research Article

Available Online through
www.ijptonline.com

FLETCHER-REEVES CONJUGATE GRADIENT NEURAL NETWORK TO SOLVE SYSTEMS OF LINEAR EQUATIONS

Dr. W. Abdul Hameed*
Department of Mathematics, School of Advanced Sciences, VIT University, Vellore – 632 014, Tamilnadu, India.
Email: hameedvellore@yahoo.co.in

Received on 06-08-2016 *Accepted on 27-08-2016*

Abstract

In this Paper, a conjugate gradient-based neural network is constructed to solve systems of simultaneous linear algebraic equations. The simulation results show that the proposed neural network is feasible and efficient for exactly determined, underdetermined and over-determined systems of linear equations. MATLAB software is used to implement the network in the computer and MATLAB coding program for the neural network has been furnished.

Key Words: Systems of Linear Equations, Neural Networks, Conjugate Gradient Learning Rule, Fletcher-Reeves Conjugate Gradient Algorithm.

1. Introduction

Several problems in Science and Engineering involve solving systems of linear algebraic equations, for example, in signal processing and robotics. In principle, solving systems of equations is equivalent to computing the inverse (or pseudo-inverse) of a matrix. We have on hand excellent numerical methods for solving systems of linear equations [1]. Typically, time constraints for solving systems of equations off-line are not important. However, if it is necessary to solve systems of equations repeatedly on line, or in real time, and if the time constraints for solving these equations are more demanding than a typical digital computer can provide, it will be necessary to find an alternative method. One approach is to use artificial neural networks because of their inherent parallel architecture [3-5]. The basic approach for solving systems of algebraic equations using neural networks involves four phases:

The first phase consists of constructing an appropriate error cost function for the particular type of problem to be solved. The error cost function is based on a defined error variable(s) which is typically formulated from a functional network for the particular problem. Thus, the problem, in general, is represented by a structured multi-layer neural network[12]. The second phase is an optimization step which involves deriving the appropriate learning rule for the structured neural network using the defined error cost function. This typically involves deriving the learning rule in its batch (or vector-matrix) form. Once the vector-matrix form for the learning rule is derived, the scalar form can be formulated in a relatively straightforward manner. The third phase involves the training of the neural network using the learning rule developed as above to match some set of desired patterns, that is, input/output signal pairs. Therefore, the network is essentially optimized to minimize the associated error cost function. That is, the training phase involves adjusting the network's synaptic weights according to the derived learning rule in order to minimize the associated error cost function. The fourth and final phase is actually the application phase in which the appropriate output signals are collected from the structural neural network for a particular set of inputs to solve a specific problem.

The structure of this Paper is as follows: In Section 2, the concept of the systems of simultaneous linear algebraic equations is presented. In Section 3, Basic Concepts, Algorithms and Architecture of Neural Network for solving the systems of linear equations are explained. In Section 4, the simulation results using MATLAB software for exactly determined, underdetermined and over-determined systems of equations are derived. Finally, the concluding remarks are listed in Section 5.

2. Systems of Simultaneous Linear Algebraic Equations

Given the set of linear algebraic equations with constant coefficients

$$\begin{array}{l} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \dots \quad \dots \quad \dots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m \end{array} \qquad (1)$$

it is desired to find the unknown quantities $x_1, x_2, \dots, x_n$, given the coefficients a_{ij}, for i = 1, 2, ..., m and j=1, 2,… , n and b_i for i =1, 2 ,.. , m. Equations (1) can be written in a more compact vector-matrix form as

$$Ax = b \qquad (2)$$

where it is assumed that $A \in \mathbf{R}^{m x n}$, $x \in \mathbf{R}^{n x 1}$, $b \in \mathbf{R}^{m x 1}$ and

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}; \quad x = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}^T; \quad b = \begin{bmatrix} b_1 & b_2 & \dots & b_m \end{bmatrix}^T$$

There are three cases that can exist: If m = n **(**there are as many equations as unknowns): The systems of equations are called exactly determined. If n > m (more unknowns than equations): The systems of equations are called underdetermined. If m > n (more equations than unknowns): This is a common situation and the systems of equations are referred to as being over-determined [7]. An efficient way to solve these systems of linear equations numerically is given by the Gauss-Jordan Elimination or by the Cholesky Decomposition [13]. For problems of the form in equation (2), where A is a singular matrix or nearly singular, the matrix A is decomposed into the product of three matrices in a process called Singular-Value Decomposition (SVD). The left and right hand matrices are left and right hand singular vectors. The middle matrix is a diagonal matrix and contains the singular values. The matrix can be inverted simply by reversing the order of the three components, transposing the singular vector matrices, and taking the reciprocal of the diagonal elements of the middle matrix. If any of the singular values is too close to zero (and therefore close to being singular), it is set to zero. The types of systems that we are here interested in to solve using neural network techniques are much more complex, large-scale over-determined and underdetermined systems, ill conditioned systems and systems that have uncertainty associated with them.

3. Basic Concepts, Algorithms and Architecture of the Neural Network Basic Concepts

The remarkable work by Warren S. McCulloch and Walter Pitts [2] in 1943 laid the foundation for the modern age neural networks. A neural network is a network of many simple processors ("units"), each possibly having a small amount of local memory. The units are connected by communication channels ("connections"), which usually carry numeric (as opposed to symbolic) data, encoded by any of various means. The units operate only on their local data and on the inputs they receive via the connections. The restriction to local operations is often relaxed during training. Some neural networks are models of biological neural networks [14] and some are not, but historically, much of the inspiration for the field of neural networks came from the desire to produce artificial systems capable of sophisticated, perhaps "intelligent", computations similar to those that the human brain routinely performs. Most neural networks have some sort of 'training" rule whereby the weights of connections are adjusted on the basis of data. In other-words, neural networks "learn" from examples, as children learn to distinguish dogs from cats based on examples of dogs and cats. If trained carefully, neural networks exhibit some capacity for generalization beyond the training data, that is, to produce approximately correct results for new cases that have not been used for training. Neural networks normally have great potential for parallelism, since the computations of the components are largely independent of each other.

Neural Network Approach

Suppose we require a solution to be found for equation (2) that is based not on a batch method but on a neural network approach that can "learn" the solution, one straightforward neural network approach is based on using the method of conjugate gradient [9].

Conjugate Gradient Learning Rule

The conjugate gradient method is developed to solve equation (2). In this method, a set of direction vectors $\{d_0, d_1,....,d_{n-1}\}$ is generated that are conjugate with respect to the matrix A, that is, $d_i^T A\, d_j = 0$ for $i \neq j$. A conjugate direction vector is generated at the kth iteration of the iterative process by adding to the calculated current negative gradient vector of the objective function, a linear combination of the previous direction vectors. This is inconsequential for the purely quadratic case. However, it is important for generalizations of the conjugate gradient method to non-quadratic problems. It is assumed that near the solution point the problem is approximately quadratic. We present here the Fletcher-Reeves conjugate gradient algorithm (with restart) [8] based on line search methods and two variations of the algorithm for non-quadratic case. Here the objective is to minimize the cost function E(x) where $x \in R^{nx1}$ and E is not necessarily a quadratic function.

Fletcher-Reeves Conjugate Gradient Algorithm

Step 1. Set x_0.

Step 2. Compute $g_0 = \nabla_x E(x_0) = \dfrac{\partial E(x_0)}{\partial x}$ at $x = x_0$

Step 3. Set $d_0 = -g_0$.

Step 4. Compute $x_{k+1} = x_k + \alpha_k d_k$, where $\alpha_k = \min\ E(x_k + \alpha d_k),\ \alpha \geq 0$

Step 5. Compute $g_{k+1} = \nabla_x E(x_{k+1})$

Step 6. Compute $d_{k+1} = -g_{k+1} + \beta_k d_k$ where $\beta_k = \dfrac{g_{k+1}{}^T g_{k+1}}{g_k{}^T g_k}$

Steps 4 through 6 are carried out for k = 0,1, …… , n-1.

Step 7. Replace x_0 by x_n and go to step 1.

Step 8. Continue until convergence is achieved; termination criterion could

be $\|d_k\| < \epsilon$ (where ϵ is an appropriate predetermined small number).

The "restart" feature in the above algorithm (step 7) is important for the cases where the cost function is not quadratic. The Fletcher-Reeves conjugate gradient algorithm is restarted by a search in the steepest descent direction after each n-iterations. The restart feature of the algorithm is important for global convergence because one cannot guarantee that the directions that d_k generate are descent directions.

Architecture of the Neural Network

By Fletcher-Reeves conjugate gradient algorithm, the update of the solution is given by $x_{k+1} = x_k + \alpha_k d_k$, where $\alpha_k = \min\ E(x_k + \alpha d_k),\ \alpha \geq 0$

The vector d_k is the current direction vector and E (.) is the cost function to be minimized. In our case the cost function is given by

$$E(x) = (1/2)\ \|Ax-b\|_2^2 = (1/2)\ (Ax-b)^T (Ax-b) \qquad (3)$$

Therefore,

$$E(x_k + \alpha\, d_k) = (1/2)\ [A(x_k + \alpha\, d_k) - b]^T [A(x_k + \alpha\, d_k) - b]$$

$$= (1/2)\ [(x_k + \alpha\, d_k)^T A^T - b^T]\ [A\, x_k + \alpha\, A\, d_k - b]$$

$$= (1/2)\ [(x_k^{\ T} + \alpha\, d_k^{\ T})\ A^T - b^T]\ [\ A\, x_k + \alpha\, A\, d_k - b\ ]$$

$$= (1/2)\ [\ x_k^{\ T} A^T + \alpha\, d_k^{\ T} A^T - b^T]\ [\ A\, x_k + \alpha\, A\, d_k - b\ ]$$

$$= (1/2)\ [\ x_k^{\ T} A^T A\, x_k + \alpha\, x_k^{\ T} A^T A\, d_k - x_k^{\ T} A^T b + \alpha\, d_k^{\ T} A^T A x_k +$$

$$\alpha^{\,2} d_k^T A^T A\, d_k - \alpha\, d_k^{\ T} A^T b - b^T A\, x_k - \alpha\, b^T A\, d_k + b^T b] \qquad (4)$$

We must compute the gradient of $E(x_k + \alpha\, d_k)$ in equation (4) w.r.t. α and set the result equal to zero.

$$\nabla_\alpha E(x_k + \alpha\, d_k) = \partial E(x_k + \alpha\, d_k) / \partial\alpha$$

$$= (1/2)\ [x_k^{\ T} A^T A\, d_k + d_k^{\ T} A^T A\, x_k + 2\,\alpha\, d_k^T A^T A\, d_k - d_k^{\ T} A^T b - b^T A\, d_k]$$

$$= (1/2)\ [2\, d_k^{\ T} A^T A\, x_k + 2\,\alpha\, d_k^T A^T A\, d_k - 2\, d_k^{\ T} A^T b]$$

$$= d_k^{\ T} A^T A\, x_k + \alpha\, d_k^T A^T A\, d_k - d_k^{\ T} A^T b = 0 \qquad (5)$$

Solving for α from equation (5) and letting $\alpha \rightarrow \alpha_k$ gives

$$\alpha_k = [\ d_k^{\ T} A^T b - d_k^{\ T} A^T A\, x_k\ ] / d_k^T A^T A\, d_k$$

$$= d_k^{\ T} [\ A^T b - A^T A\, x_k\ ] / d_k^T A^T A\, d_k$$

$$= - d_k^{\ T} g_k / d_k^T A^T A\, d_{k,} \quad \text{where } g_k = A^T A\, x_k - A^T b$$

$$= - g_k^{\ T} d_k / d_k^T A^T A\, d_k \qquad (6)$$

Here g_k is the gradient of $E(x_k)$.

That is, $g_k = \nabla_x E(x_k) = A^T A\, x_k - A^T b$ (7)

4. Simulation

In order to verify the feasibility and efficiency of the above discrete-time neural network, three simple examples are carried out using MATLAB software [10] to solve exactly determined, underdetermined and over-determined systems of equations.

Example-1: Consider the following exactly determined linear equations taken from [11].

$$\begin{aligned} 10x_1-2x_2-x_3-x_4 &= 3 \\ -2x_1+10x_2-x_3-x_4 &= 15 \\ -x_1-x_2+10x_3-2x_4 &= 27 \\ -x_1-x_2-2x_3+10x_4 &= -9 \end{aligned} \qquad (8)$$

The discrete-time conjugate gradient algorithm given above is used to solve this system for **x**. We assume 'zero' initial condition for the unknown quantities. After the 4th iteration, the solution is given as x = [1, 2, 3, 0]. The same solution was obtained using the Gauss-Seidel method after 7 iterations and Jacobi's method after 12 iterations. Figure (1) shows the convergence profile for the unknowns.The coding in MATLAB for the above discrete neural network for this problem is given in Figure (4).

Example – 2: Consider the following underdetermined systems of linear equations taken from [8].

$$\begin{aligned}
6x_1+2x_2+4x_3-9x_4-12x_5+2x_6-12x_7+x_9 &= -12\\
8x_1-10x_2+x_3+8x_4-22x_5-11x_7-11x_8+7x_9 &= -13\\
9x_1-7x_2-6x_3+6x_4+10x_5-10x_6+15x_7-13x_8-12x_9 &= 9\\
-10x_1+11x_2-6x_3-8x_4-5x_5-9x_6+x_7-3x_8-5x_9 &= 0\\
2x_1-x_2+4x_3-3x_4+3x_5-4x_6-12x_7+10x_8-3x_9 &= -6
\end{aligned}$$

$$A=\begin{pmatrix} 6 & 2 & 4 & -9 & -12 & 2 & -12 & 0 & 1\\ 8 & -10 & 1 & 8 & -22 & 0 & -11 & -11 & 7\\ 9 & -7 & -6 & 6 & 10 & -10 & 15 & -13 & -12\\ -10 & 11 & -6 & -8 & -5 & -9 & 1 & -3 & -5\\ 2 & -1 & 4 & -3 & 3 & -4 & -12 & 10 & -3 \end{pmatrix}$$

$$x=\begin{bmatrix}x_1 & x_2 & \dots & x_9\end{bmatrix}^T; \qquad b=\begin{bmatrix}-12 & -13 & 9 & 0 & -6\end{bmatrix}^T;$$

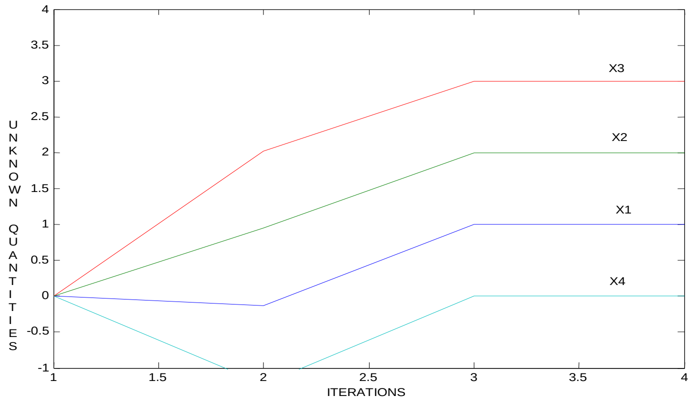
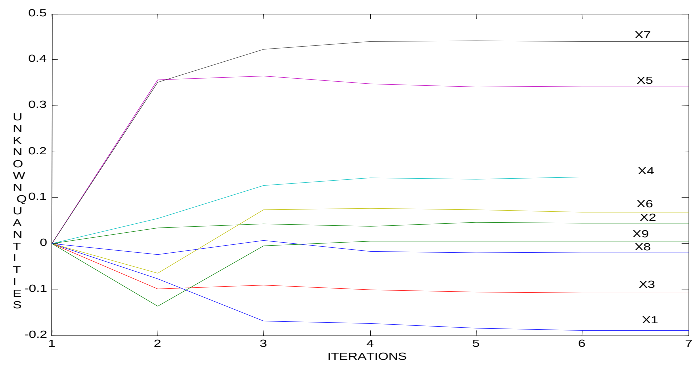


Figure (1): Convergence profile for exactly determined systems.

Zero initial condition is assumed for **x**. The figure (2) shows the convergence profile for the unknown quantities. As can be seen, the neural network converges after the 5th iteration and the solution is given as

$$x=\begin{bmatrix}-0.1885 & 0.0443 & -0.1064 & 0.1449 & 0.3417 & 0.0677 & 0.4391 & -0.0186 & 0.0056\end{bmatrix}$$

The same solution was obtained using the Singular-Value Decomposition (SVD) method.



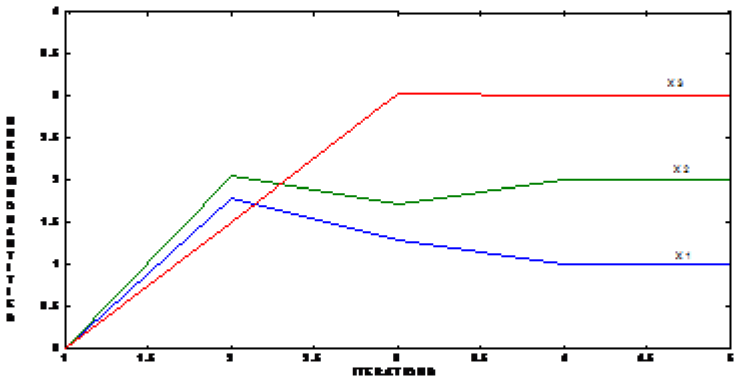
Figure(2): Convergence profile for underdetermined systems.

Example-3: Consider the following over-determined systems of linear equations taken from [6].

$$
\begin{aligned}
x_1+2x_2+3x_3 &= 14\\
3x_1+2x_2+x_3 &= 10\\
x_1+x_2+x_3 &= 6\\
2x_1+3x_2-x_3 &= 5\\
x1 + \quad x2 &= 3
\end{aligned}
$$

Here $A=\begin{pmatrix} 1 & 2 & 3\\ 3 & 2 & 1\\ 1 & 1 & 1\\ 2 & 3 & \text{-}1\\ 1 & 1 & 0\end{pmatrix}$; $\qquad b=\begin{bmatrix}14 & 10 & 6 & 5 & 3\end{bmatrix}^T$

Zero initial condition is assumed for the unknown quantities. The Figure (3) shows the convergence profile for the unknown quantities. As can be seen, the neural network converges after the 3rd iteration and the solution is given as $x=\begin{bmatrix}1 & 2 & 3\end{bmatrix}^T$.



Figure(3) : Convergence profile for over-determined system.

MATLAB coding to solve equations (8)

% Solution of (exactly determined) systems of linear equations

% using neural network and MATLAB

% t=Number of iterations

```
% n=Number of unknowns

A=[10 -2 -1 -1;-2 10 -1 -1;-1 -1 10 -2;-1 -1 -2 10];  b=[3;15;27;-9];

x=[0;0;0;0];  t=4;  n=4;

g=A'*A*x-A'*b;  d=-g;

for k=1:t

x1(k,1)=x(1,1); x2(k,1)=x(2,1); x3(k,1)=x(3,1); x4(k,1)=x(4,1);

q=-(g'*d)/(d'*A'*A*d);

x=x+q*d;

g1=A'*A*x-A'*b;

p=(g1'*g1)/(g'*g);

d=-g1+p*d;

g=g1;

end

x

pause

k=1:t;

plot(k,x1(k,1),k,x2(k,1),k,x3(k,1),k,x4(k,1))

axis([1  t  -1  4])

xlabel('ITERATIONS')

ylabel('UNKNOWN QUANTITIES')

gtext('X1'),   gtext('X2'),   gtext('X3'),  gtext('X4')
```

5. Conclusion

The main purpose of this Paper is to introduce the concept of neural network and to develop the neural algorithm using the conjugate gradient rule to solve the systems of linear equations for all the three cases, namely exactly determined, underdetermined and over-determined.

The neural algorithm proposed in this Paper has several advantages over the conventional parallel algorithms. First, the neural algorithms are much simpler than others. The only algebraic operations required in the neural algorithms are addition and multiplication. Inverse and other complicated logic operations are not needed. Second, the neural algorithms are the most parallel, compared with conventional parallel algorithms. Simplicity in implementation of the neural algorithms is the third advantage.

We have presented the MATLAB programming code and graph to understand the updating process of the neural network architecture and convergence profile of the unknowns.

References

1. E. Anderson, Z. Bai, C.H. Bischof, J.W. Demmel,J.J. Dongarra, J.J. Du Croz, A. Greenbaum, S.J. Hammarling, A. McKenney, S. Ostrouchov, and D.C. Sorenson, *LAPACK Users' Guide,* Release 2.0**,** 2nd ed., Philadelpha: Society for Industrial and Applied Mathematics, (1995).
2. J.A. Anderson, and E. Rosenfield, *Neurocomputing: Foundations of Research,* Cambridge, MA:M.I.T. Press, pp. 18-27,(1998).
3. A. Cichocki, and R. Unbehauen, *Neural Network for Solving Systems of Linear Equations and Related Problems*, IEEE Transactions on Circuits and Systems, vol. CAS-39, (1992); pp. 124-38.
4. A. Cichocki, and R. Unbehauen, *Neural Network for Solving Systems of Linear Equations – Part II: Minimax and Least Absolute Value Problems*, IEEE Transactions on Circuits and Systems, vol.CAS-39, (1992); pp. 619-33.
5. A. Cichocki, and R. Unbehauen, *Neural Network for Optimization and Signal Processing,* Wiley, New York, (1993).
6. H.K. Dass, H.C. Saxena, and M.D. Raisinghania, *Simplified Course in Matrices*, S. Chand & Co. Ltd., New Delhi, (1999).
7. G.H.Gulub, and C.F. Van Loan, *Matrix Computations,* 3rd ed., Baltimore, MD: Johns Hopkins University Press, (1996).
8. F.M. Ham, and Ivica Kostanic, *Principles of Neurocomputing for Science and Engineering*, Tata McGraw-Hill Pub. Co. Ltd., New Delhi, 2nd ed., pp:350-385, (2003).
9. S.G. Nash, and A. Sofer, *Linear and Nonlinear Programming,* McGraw-Hill, New York,(1996).
10. Rudra Pratap ,*Getting Started with MATLAB – A quick Introduction for Scientists and Engineers,* version –6, Oxford University Press, New York, (2003).
11. S.S. Sastry, *Introductory methods of numerical analysis*, Prentice- Hall of India Pvt. Ltd., New Delhi, pp:112, (1988).
12. L.X. Wang, and J.M. Mendel, *Structured Trainable Networks for Matrix Algebra*, Proceedings of the International Joint Conference on Neural Networks, San Diego, CA, vol. 2, (1990); pp. 125-32.

13. D.S. Watkins, *Fundamentals of Matrix Computations,* Wiley, New York, (1991).

14. B. Widrow, and M.A. Lehr, *30 Years of Neural Networks: Percetron, Madaline and Backpropagation*, Proceedings of the IEEE, vol. 78, (1990); pp.1415-42.

Corresponding Author:

Dr. W. Abdul Hameed*

Email: *hameedvellore@yahoo.co.in*