Solving Systems of Linear Equations and Finding the Inversion of a Matrix by Neural Network Using Genetic Algorithms (NN using GA)

Z. Ghassabi, B. Moaveni, A. Khaki-Sedigh
Department of Computer Science
Islamic Azad University, Science and Research Branch
Tehran
IRAN
rose_z60gh@yahoo.com b_moaveni@alborz.kntu.ac.ir sedigh@kntu.ac.ir

Abstract: - In this paper, we propose a one-layered neural network that recovers its input variables by genetic algorithms to solve the systems of linear equations (or, equivalently, matrix inversion). First we described solving systems of linear equations (matrix inversion) by mentioned neural network. Then, experimental results are presented to show the effectiveness of the approach. Finally, future avenue of this research is proposed.

Key-Words: - Solving Systems of Linear Equations, Neural Network, Genetic Algorithms, Matrix Inversion, Block of the Recovery.

1 Introduction

Many problems in science and engineering (e.g. robotics and signal processing) require solving systems of linear equations (matrix inversion). When solving systems of linear equations offline with numerical methods, the goal is to find optimal solution, without considering a time constraint. If such a problem needs to be solved in real-time (e.g. controller) under time constraints, existing numerical methods may not scale well.

Well-known techniques to solve systems of linear equations include:

Graphing, Substitution, Elimination/Addition, Gauss_Jordan Elimination, Cramer's Rule, Matrix Algebra /Inverses, …

Of these techniques, Gauss_Jordan Elimination, Cramer's Rule, Matrix Inverses apply only to linear systems of equations, subject to determinant of the coefficient matrix not being zero. Numerical methods are not universally applicable and cannot guarantee that all solutions of systems of linear equations have been found.

solving systems of Linear equations has been studied in the field of neural networks [5], [6] with varying results.

In this paper, a one-layered neural network is proposed to solve the systems of linear equations that it is a simple structure of feedforward neural networks (FNN) and it recovers its inputs by genetic algorithms.

The rest of the paper is organized as follows: Section 2 presents problem formulation, while the neural network model and recovering of the inputs in mentioned neural network are presented in section 3 with experimental results discussed in section 5.1. Section 4 uses the mentioned NN for finding the inversion of a matrix with experimental results discussed in section 5.2.

2 Problem formulation to solve the Systems of Linear Equations

Assume that a given n order system of linear equation:

$$\begin{aligned} &a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ &a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ &\vdots \\ &a_{21}x_1 + a_{12}x_2 + \ldots + a_{nn}x_n = b_n \end{aligned} \tag{1}$$

which can be simplified into $Ax = b$, and

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ \vdots & & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}_{n*n}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}_{n*1}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}_{n\times 1}$$

It is required to construct the algorithm finding for any matrix A any value of the vector x satisfying all the linear equations simultaneously.

In the feedforward neural network (FNN) with one layer, the input vector are related to the output layer as follows:

$$Y = f(pW - b) \tag{2}$$

Equation (1) can be rewritten as:

$$pW = b$$

$$\sum_{j=1}^{n}\sum_{i=1}^{n} W_{ji}\, p_j = \sum_{j=1}^{n} b_j \tag{3}$$

So the vector x consisting of n components will be an input of a neural network (NN).
From the above analyses, we wish to design a neural network to solve the systems of linear equations. In the following section, we will present the structure of neural network and recovery block to solve the systems of linear equations.

3 The Neural Network Model for Solving Systems of Linear Equations

3.1 The FNN Model for Solving Systems of Linear Equations

As mentioned in previous section, the network for computing the solution of systems of linear equations is a one-layered FNN, with the n input variables as $p_1, p_2, ..., p_n$, vector $(bias)b$ of desire outputs (supervising learning rule) and the matrix of weighting coefficients A. The activation function in equation (2) is linear (Fig.1).

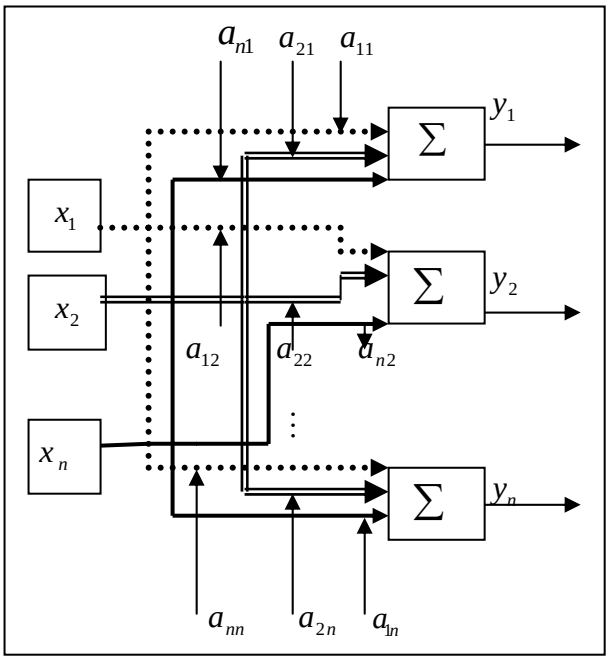

Fig.1 Neural Network model for equations (1) and (3).

The error for each output neuron in NN will be determined as follows:

$$e_i = b_i - y_i \tag{4}$$

where,

$$y_i = \sum_{j=1}^{n} p_j W_{ij}$$

In the network, input vector, p, is going to close to vector x ($\min\|x - p\|$) by iterating procedure of the input recovery as the error function minimizes. Therefore, vector p is the solution of equation (1). The diagram of NN with the block of the recovery which uses GA to solve the systems of linear equations is shown in Fig.2.

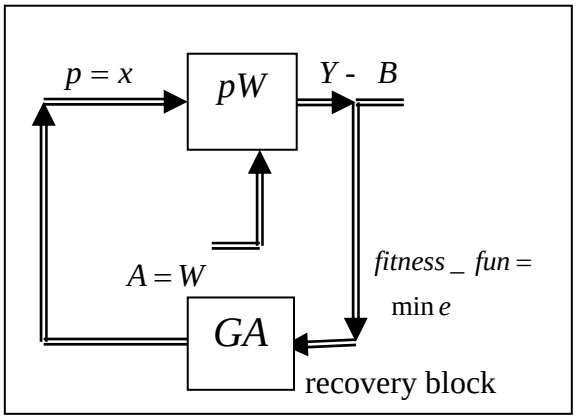

Fig.2 Neural Network structure which uses genetic algorithm for recovering of inputs.

3.2 Recovery Block for recovering of NN's inputs for Solving Systems of Linear Equations

Genetic Algorithms search from a populations of solutions, use fitness function and probabilistic transition rules.
Let $P(t)$ and $C(t)$ be parents and offspring in a current generation t; the general structure of genetic algorithms is described as follows:

Procedure: Genetic Algorithms
begin
 Initialize $P(t)$;
 evaluate $P(t)$;
 while (not termination condition) **do**
  recombine $P(t)$ to yield $C(t)$;
  (crossover, mutation)
  evaluate $C(t)$;
  select $P(t+1)$ from $P(t)$ and $C(t)$;
  $t = t + 1$
 end
end

For matrix $A_{n\times n}$ in linear matrix equations, $Ax = b$, steps of writing GA recovery block is as follows:

1-determining the values of population in the first generation for genetic algorithm, mutation rate (denoted by P_m), crossover rate (denoted by P_c), the number of genes, the population size and the maximum iteration.
According to variables boundaries, initial population is randomly generated. Each chromosome consists of n (the number of components in vector x) genes.
2-Determining fitness function and mutation, crossover operations
We want to minimize error function in the neural network. Therefore, we choose some chromosomes which have minimum fitness function value.
The fitness function from equations (1), (4) is as follows:

$$fitness\_function = (|e_1| + |e_2| + \ldots |e_n|) \quad (5)$$

The arithmetic crossover is defined as the combination of two chromosomes Parent1 and Parent2 as follows [8]:

$$\begin{aligned} &\text{Child1} = \text{Parent1} + (1 - \text{rand})\text{Parent2} \\ &\text{Child2} = \text{Parent2} + (1 - \text{rand})\text{Parent1} \end{aligned} \quad (6)$$

where rand is a random number and $\text{rand} \in (0,1)$. Child1 and Child2 are the resulted chromosomes after the crossover operations.
The non-uniform mutation is given as follows [8]: for a given parent $V = [g_1, \ldots, g_k, \ldots, g_n]$ if the element g_k of it is selected for mutation, the resulting offspring is $V' = [g_1, \ldots, g_k', \ldots, g_n]$, where g_k' is randomly selected from two possible choices:

$$\begin{aligned} &g_k' = g_k + \Delta(t, g_k^{\text{Upper}} - g_k) \\ &\text{or } g_k' = g_k - \Delta(t, g_k - g_k^{\text{Lower}}) \end{aligned} \quad (7)$$

where $g_k^{\text{Upper}}, g_k^{\text{Lower}}$ are the upper and lower bounds for g_k. The function $\Delta(t, y)$ returns a value in the range $[0, y]$ such that the value of $\Delta(t, y)$ approaches to 0 as t increases (t is the generation number) as follows:

$$\Delta(t, y) = y.r.(1 - \frac{t}{T})^b \quad (8)$$

where, r is a random number from $[0,1]$, T is the maximal generations number and b is a parameter determining the degree of non-uniformity.

4 Finding the Inversion of a Matrix

Finding the inversion of matrix A is the same as solving the systems of linear equations as follows:

$$AB = I \quad (9)$$

that

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ \vdots & & & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}_{n*n}, B = \begin{bmatrix} b_{11}, b_{12}, \ldots, b_{nn} \\ b_{21}, b_{22}, \ldots, b_{2n} \\ : \\ b_{n1}, b_{n2}, \ldots, b_{nn} \end{bmatrix}_{n \times n}, \; b = \begin{bmatrix} 1,0,\ldots,0_1 \\ 0,1,\ldots,0 \\ : \\ 0,0,\ldots,1 \end{bmatrix}_{n \times n}$$

I, B are identity matrix and inversion of matrix A respectively ($AA^{-1} = I$).
We rewrite equation (9) as follows:

$$AW = \mathrm{I} \quad (10)$$

Matrix W is the solution of the system of linear equations (9), (10) and input variable in neural network in Fig.1.
For finding the inversion of matrix $A_{n \times n}$, the neural network block in Fig.2 is used n times for recovering the components of matrix W resulted by recovery block. On the other hand we have n NN that each one consists of the matrix of weighting coefficients A, j th column of B as input variables and corresponding column from I as desire outputs.
The fitness function from equation (9) is as follows:

$$fitness\_function = (|E_1| + |E_2| + \ldots |E_n|) \quad (11)$$

that E_i, $i = 1 \ldots n$ is the value of error in i th NN. Error for each NN is computed by equation (5).

5 Experimental results

5.1 Experiments for Solving Systems of Linear Equations

Assume $A_{3\times3}$, $b_{1\times3}$ as follows:

$$A = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 0.5 \\ -1 & 0 & 4 \end{bmatrix}_{3\times3}, x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}_{3\times1}, b = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix}_{3\times1}$$

that $A \neq A^T$ and the elements of A can be negative number. The initial values for computer program are as follows:

P_m	P_c	The number of genes	Maximum population	Maximum Generation
0.6	0.9	3	30	350

Initial genes were generated randomly as $[-0.5, 0.5]$. The problem solution computed by our proposed approach is as follows:

$$x_1 = 1.1739, \; x_2 = -0.3478, \; x_3 = 1.0435$$

Let us consider the following ill-conditioned matrix $A_{3\times5}$:

$$A=\begin{bmatrix}2 & -2 & 1 & 1 & 0.5\\ 1 & 1 & 1 & 0 & 0.5\\ 3 & -1 & 2 & 1 & 1\end{bmatrix}_{3\times3},\ x=\begin{bmatrix}x_1\\ x_2\\ x_3\\ x_4\\ x_5\end{bmatrix}_{3\times1},\ b=\begin{bmatrix}2\\ 1\\ 3\end{bmatrix}_{3\times1}$$

The final solution (equilibrium point) obtained by using our program was

$$\bar{x}^{*}=[0.5065\quad -0.0934\quad 0.3369\quad 0.2134\quad 0.5000]^{T}$$

$$A\bar{x}^{*}=[2.0000\quad 1.0000\quad 3.0000]^{T}$$

which is excellent agreement with the exact minimal norm solution

$$x^{*}=[0.5748\quad -0.0866\quad 0.4094\quad 0.1654\quad 0.2047]^{T}$$

where, T denotes the transpose of a vector or matrix, obtained by using command $\mathrm{pinv(A)*b}$ in MATLAB7. The initial values for computer program are as follows:

P_m	P_c	The number of genes	Maximum population	Maximum Generation
0.6	0.9	5	50	400

5.2 Experiments for finding the inversion of a Matrix

Assume $A_{2\times2}$ as follows:

$$A=\begin{bmatrix}2 & 2\\ 2 & 4\end{bmatrix}_{2\times2},\ B=\begin{bmatrix}b_{11},b_{12}\\ b_{21},b_{22}\end{bmatrix}_{2\times2},\ I=\begin{bmatrix}1,0\\ 0,1\end{bmatrix}_{2\times2}$$

The initial values for computer program are as follows:

P_m	P_c	The number of genes	Maximum population	Maximum Generation
0.6	0.9	4	50	300

Initial genes were generated randomly from $[-0.5,0.5]$. The solution by runnig of our computer program is as follows:

$$\bar{B}^{*}=\begin{bmatrix}0.9997 & -0.5000\\ -0.4999 & 0.5000\end{bmatrix}$$

The solution by using numerical methods is as follows:

$$\mathrm{B}^{*}=\begin{bmatrix}1.0000 & -0.5000\\ -0.5000 & 0.5000\end{bmatrix}$$

6 Conclusion

Discussed approach for solving systems of linear equations and finding the inversion of a matrix is shown by experiments to be very effective and feasible method. The fast training speed and high accuracy are the most important advantages for this method. In this paper, Genetic algorithm provided us a great flexibility to hybridize with NN to make an efficient implementation for solving systems of linear equations and finding the inversion of a matrix. Hybridizing GA with other artificial algorithms for solving systems of linear equations with multiple solutions is future avenue of this research.

References:

[1] Andries P.Engelbercht, *Computational Intelligence*, University of Pretoria, South Africa, 2002

[2] Mitsuo Gen,Runwei Cheng, *Genetic Algorithms and engineering design*, Ashikaga Institute of Technology Ashikaga, Japan, 1997

[3] A. I. Galushkin, V.A. Sudarikov, Adaptive Neural algorithm for Solving Linear Algebra Problems, *IEEE*, Vol.1, 1992, pp. 128-138.

[4] R. Brits, A. P. Engelbrecht, F. van den Bergh, Solving Systems of Unconstrained Equations using Particle Swarm Optimization, *Systems, Man and Cybernetics, 2002 IEEE International Conference*, Vol.3, 2002, Page(s):6 pp.

[5] A. Cichocki and R. Unbehaun, Neural Networks for Solving Systems of Linear Equations And Related Problems, *IEEE Transactions on Circuts and Systems-I. Fundemental Theory and Applications*. Vol 39, No. 2, February 1992, pp.124-137.

[6] D. S. Huang and Z. Chi, Solving Linear Simultaneous Equations by Constraining Learning Neural Networks, *International Joint Conference on Neural Networks*, Vol Addendum, Praque, 1995, pp. 775-779.

[7] M. B. Menhaj, *Computational intelligence*, Amirkabir University, Iran, 2002

[8] Z. Michalewicz, *Genetic Algorithms + Data Structure= Evolution Programs*, 2nd ed., Springer- Verlag, New York, 1994.