Available online at http://ijim.srbiau.ac.ir/
Int. J. Industrial Mathematics (ISSN 2008-5621)
Vol. 7, No. 1, 2015 Article ID IJIM-00479, 9 pages
Research Article

On the convergence speed of artificial neural networks in the solving of linear systems

A. Jafarian [*][†]

Abstract

Artificial neural networks have the advantages such as learning, adaptation, fault-tolerance, parallelism and generalization. This paper is a scrutiny on the application of diverse learning methods in speed of convergence in neural networks. For this aim, first we introduce a perceptron method based on artificial neural networks which has been applied for solving a non-singular system of linear equations. Next two famous learning techniques namely, the steepest descent and quasi-Newton methods are employed to adjust connection weights of the neural net. The main aim of this study is to compare ability and efficacy of the techniques in speed of convergence of the present neural net. Finally, we illustrate our results on some numerical examples with computer simulations.

Keywords : System of linear equations; Quasi-Newton method; Steepest descent method; Cost function; Learning algorithm.

1 Introduction

As a matter of fact, it might be said that numerical solutions of almost all practical engineering and applied science problems routinely require solution of a linear system problem. Hence various methods for solving this problem have been proposed. One indirect scheme is using artificial neural network approach. Neural networks (NNs) are nonlinear mapping structures based on the function of the human brain. The study of brain-style computation has its roots over 50 years ago in the work of McCulloch and Pitts (1943) [4] and slightly later in Hebbos famous Organization of Behavior (1949) [3]. The next work in artificial intelligence was torn between those who believed that intelligent systems could best be built on computers modeled after brains, and those like Minsky and Papert (1969) [5] who believed that intelligence was fundamentally symbol processing of the kind readily modelled on the von Neumann computer. For a variety of reasons, the symbol-processing approach became the dominant theme in artificial intelligence. Hopfield (1985) provided the mathematical foundation for understanding the dynamics of an important class of networks. Rumelhart and McClelland (1986) introduced the back-propagation learning algorithm for complex, multi-layer networks and thereby provided an answer to one of the most severe criticisms of the original perceptron work [2]. Neural nets are powerful tools for modelling, specially when the underlying data relationship is unknown. There are several types of architecture of neural networks. The two most widely used NN are the feed-forward and feed-back neural nets. Multi-layer feed-forward neural network or multi-layer perceptron is very popular and is used more than other neural network type for a wide variety of

[*]Corresponding author. Jafarian5594@yahoo.com
[†]Department of Mathematics, Urmia Branch, Islamic Azad University, Urmia, Iran.

tasks.
In this paper, we first introduce a type of architecture of NN which has been offered to find approximate solution of the linear system $AX = B$ arise in a wide variety of applications. We discuss methods only for non-singular linear systems in this study. The considered perceptron NN is a two-layer feed-forward networks with n neurons in input layer and one neuron in output layer. For the given neural net, the $n \times n$ coefficients matrix A and the n-vector B are considered as training patterns. Outputs from the NN which are also a real number, is numerically calculated for real weights and real inputs and then are compared with target output. Next a cost function is defined that measures the difference between the target output and corresponding calculated output. Then the suggested neural net by using the steepest descent and quasi-Newton methods that are two well known learning techniques, adjusts the real connection weights to any desired degree of accuracy, respectively. Finally, amount of influence for each techniques in speed of convergence are compared together. This paper is organized as follows. First in Section 2, the basic definitions used of linear systems are briefly presented. Section 3 describes how to apply the given techniques for approximating solution of the linear system. Also the present techniques are compared in this section. Finally, some examples are collected in Section 4.

2 Preliminaries

In this section we give a detailed study of artificial neural networks which are used in the next sections.

Definition 2.1 *Artificial neural systems or neural networks are physical cellular systems which can acquire, store and utilize experiential knowledge [8].*

Neural networks can be considered as simplified mathematical models of brain and they function as parallel distributed computing networks.

Definition 2.2 *A feed-forward neural net is a type of architecture of artificial neural networks such that allows signals to travel one way only. In other word the output of any layer does not affect that same layer or previous layers [1].*

2.1 Input-output relation of each unit

We have given a short review on learning of feed-forward neural networks with real set input signals and real number weights (FNN) [8]. Consider a two-layer FNN with n input units and one output unit. When a real input vector $A_p = (a_{p1}, a_{p2}, ..., a_{pn})$ is presented to our FNN, then the input-output relation of each unit can be written as follows:

Input units:
The input neurons make no change in their inputs. So,

$$O_{pj} = a_{pj}. \tag{2.1}$$

Output unit:

$$Y_p = f(Net_P),$$

where

$$Net_P = \sum_{j=1}^{n} w_j.O_{pj}, \;\; p, j = 1, 2, ..., n. \tag{2.2}$$

where $a_{p1}, ..., a_{pn}$ are real numbers and w_j is real connection weight (see Fig. 1).

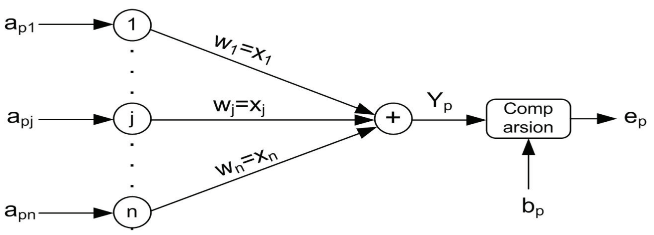

Figure 1: The proposed FNN.

3 Linear system of equations

In this section we concentrate on finding approximate solution for the non-homogeneous system of n equations in n unknowns in form

$$\begin{cases} a_{11}x_1 + ... + a_{1n}x_n = b_1 \\ \vdots \\ a_{i1}x_1 + ... + a_{in}x_n = b_i \\ \vdots \\ a_{n1}x_1 + ... + a_{nn}x_n = b_n \end{cases}, \tag{3.3}$$

where $a_{i,j}$, b_i, x_j $(for\ i,j = 1,...,n)$ are real numbers. The matrix form of the present system is

$$AX = B,$$

where

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{bmatrix}, \; X = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix},$$

$$B = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}, \; b \neq 0.$$

We assume that, given $n \times n$ matrix A and n-vector B the system is consistent.
Now consider again the two-layer neural network that has been showed in Fig. 1. Each equation in system (3.3) can be interpreted as a training pattern for the present neural net, where the antecedent and the consequence parts of the equations are the input signals and the desired output of the neural net, respectively. Consequently, the training set derived from above system can be written in the form

$$\{A_p; b_p\} = \{(a_{p1}, ..., a_{pn}); b_p\}, \; p = 1, ..., n.$$

3.1 Cost function

Let real quantities w_j $(for\ j = 1,...,n)$ are initialized at random values for parameters x_j. The actual output of the FNN can be calculated by using Eqs. (2.1)-(2.2). Now we want to define a cost function that measures the difference between the target output and the corresponding actual output. The error function for the $p-th$ training pattern is defined by

$$e_p = \frac{(b_p - Y_p)^2}{2}. \tag{3.4}$$

The cost function for the input-output pair $\{A_P; b_p\}$ is obtained as:

$$e = \sum_{p=1}^{n} e_p. \tag{3.5}$$

Theoretically this cost function satisfies the following equation if we use infinite number of training patterns,

$$e_p \longrightarrow 0 \Longleftrightarrow Y_p \longrightarrow b_p.$$

3.2 Learning algorithm of the FNN

The main aim of this section is to introduce how to deduce a learning algorithm to update the real weights w_j $(for\ j = 1,...,n)$. For this scope in continuance, we suggest two famous learning techniques namely, the steepest descent and quasi-Newton methods. By using these techniques two different learning rules are obtained in which help us to update the connection weights in the present neural net to any degree of accuracy.

3.2.1 The steepest descent method

In the subsection, a method will be considered which makes use of the gradient of the function $e_p(w_1, ..., w_n)$ which is evaluated at the point w_j as:

$$g_j = \frac{\partial e_P}{w_j}, \; j = 1, ..., n. \tag{3.6}$$

Descent methods all use the basic iterative step

$$w' = w - \eta B g, \tag{3.7}$$

where B is a matrix defining a transformation of the gradient and η is a step length [6]. The simplest such algorithm the method of steepest descents, was proposed by Cauchy (1848) for the solution of systems of nonlinear equations with $B = I_n$. we intend to use this method for updating the connection weights in the FNN. Consequently, for real parameter w_j adjust rule can be written as follows:

$$w_j(t+1) = w_j(t) + \Delta w_j(t). \tag{3.8}$$

Also we can derive the amount of adjustment for each parameter w_j as follows:

$$\Delta w_j(t) = -\eta \frac{\partial e_p}{\partial w_j} + \alpha \Delta w_j(t-1), \tag{3.9}$$

where t indexes the number of adjustments, η is the learning rate and α is the momentum term constant. Thus our problem is to calculate the derivative $\frac{\partial e_p}{\partial w_j}$ in (3.9). The derivative can be calculated from the cost function e_p by using the input-output relations of the FNN. We calculate this derivative as follows:

$$\frac{\partial e_p}{\partial w_j} = \frac{\partial e_p}{\partial Y_p} \cdot \frac{\partial Y_p}{\partial Net_p} \cdot \frac{\partial Net_p}{\partial w_j}, \; j = 1, ..., n. \tag{3.10}$$

Consequently,

$$\frac{\partial e_p}{\partial w_j} = -a_{pj}.(b_p - Y_p),\ p = 1, ..., n. \tag{3.11}$$

Finally, weight adjustments are summarized in the supervised mode as follows:

$$w_j(t+1) = w_j(t) + \eta.a_{pj}.(b_p - Y_p) + \alpha.\Delta w_j(t-1). \tag{3.12}$$

The learning stops when the weight vector $(w_1, ..., w_n)$ remains unchanged during a complete training cycle. Now let us assume that input-output pair $\{A_p; b_p\}$ where $A_p = (a_{p1}, ..., a_{pn})$ are given as training data. Then the learning algorithm can be summarized as follows:

Learning algorithm

Step 1: $\eta > 0$, $\alpha > 0$, $Emax > 0$ are chosen. Then the real vector $(w_1, ..., w_n)$ are initialized at random values.
Step 2: Let $t := 0$ where t is number of iterations of the learning algorithm. Then the running error E is set to 0.
Step 3: Let $t := t + 1$. Repeat the following procedures for $p = 1, ..., n$:

i Forward calculation: Calculate the actual output Y_p by presenting the input vector A_p.

ii Back-propagation: Adjust crisp parameter w_j by using the cost function (3.5) and the adjustment relation (3.12).

Step 4: Cumulative cycle error is computed by adding the present error to E.
Step 5: The training cycle is completed. For $E < Emax$ terminate the training session. If $E > Emax$ then E is set to 0 and we initiate a new training cycle by going back to *Step 3*.

3.2.2 The quasi-Newton method

Variable metric algorithms also called quasi-Newton or matrix iteration algorithms, have proved to be the most effective class of general-purpose methods for solving unconstrained minimization problems. Their development is continuing so rapidly, however, that the vast array of possibilities open to a programmer wishing to implement such a method is daunting. Here an attempt will be made to outline the underlying structure on which such methods are based. This subsection describes one set of choices of strategy for approximating solution of the given linear system [6].

Definition 3.1 *The hessian matrix H associated with given a continuously differentiable function $F : R^n \longrightarrow R^n$ at the point $x = (x_1, ..., x_n)^T$ is defined as:*

$$H(x) = (h_{ij}(x))_{n\times n},$$

where

$$h_{ij}(x) = \frac{\partial^2 f(x)}{\partial x_i \partial x_j},\ i, j = 1, ..., n.$$

In the quasi-Newton method for the present function $F : R^n \longrightarrow R^n$ and an initial vector $x^{(0)} \epsilon R^n$ at each step t one has to accomplish the following operations [7]:

i Compute the hessian matrix $H_t = H(x^{(t)})$.

ii Find a descent direction $d^{(t)}$ as follows:

$$d^{(t)} = \nabla F(x^{(t)}) = [\frac{\partial F}{\partial x_1}, \cdots, \frac{\partial F}{\partial x_n}]^T_{x=x^{(t)}}.$$

iii Compute the acceleration parameter μ_t as:

$$\mu_t = \frac{(d^{(t)})^T d^{(t)}}{(d^{(t)})^T H_t\ d^{(t)}}.$$

iv Update the solution vector $x^{(t)}$ as following:

$$x^{(t+1)} = x^{(t)} + \Delta x^{(t)}, \tag{3.13}$$

where

$$\Delta x^{(t)} = -\mu_t.d^{(t)} + \alpha.\Delta x^{(t-1)}.$$

In continuance, we want to utilize the quasi-Newton method for updating the connection weight w_j. Let us rewrite the cost function (3.5) as follows:

$$e = \sum_{j=1}^{n} \frac{1}{2}\ (b_j - \sum_{i=1}^{n}(a_{ji}.w_i))^2 =$$

$$\frac{1}{2}\sum_{j=1}^{n}(b_j - \sum_{i=1}^{n}(a_{ji}.w_i))^2 =$$

$$\frac{1}{2}\sum_{j=1}^{n}\{b_j^2 - 2b_j.(\sum_{i=1}^{n} a_{ji}.w_i) + (\sum_{i=1}^{n} a_{ji}.w_i)^2\} =$$

$$\frac{1}{2}\sum_{j=1}^{n} b_j^2 - \sum_{j=1}^{n}\sum_{i=1}^{n}(b_j.a_{ji}.w_i) + \frac{1}{2}\sum_{j=1}^{n}(\sum_{i=1}^{n} a_{ji}.w_i)^2$$

$$= \frac{1}{2}\sum_{j=1}^{n} b_j^2 - [\sum_{j=1}^{n}(b_j.a_{j1}), \quad \dots \quad ,\sum_{j=1}^{n}(b_j.a_{jn})] \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}$$

$$+\frac{1}{2}\begin{bmatrix} w_1, & \dots & ,w_n \end{bmatrix} \begin{bmatrix} \sum_{j=1}^{n} a_{j1}^2 & \dots & \sum_{j=1}^{n} a_{j1}a_{jn} \\ \sum_{j=1}^{n} a_{j2}a_{j1} & \dots & \sum_{j=1}^{n} a_{j2}a_{jn} \\ \vdots & \ddots & \vdots \\ \sum_{j=1}^{n} a_{jn}a_{j1} & \dots & \sum_{j=1}^{n} a_{jn}^2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}.$$

For simplify, the cost function e can be summarized as following:

$$e = C - B^T W + \frac{1}{2} W^T Q W, \tag{3.14}$$

where

$$Q = (q_{ij})_{n\times n}, \; q_{ij} = q_{ji} = \sum_{k=1}^{n} a_{ki}.a_{kj},$$

$$C = \frac{1}{2}\sum_{j=1}^{n} b_j^2,$$

$$B^T = [\sum_{j=1}^{n}(b_j.a_{j1}), \quad \dots \quad ,\sum_{j=1}^{n}(b_j.a_{jn})],$$

$$W^T = \begin{bmatrix} w_1, & \dots & ,w_n \end{bmatrix}.$$

Clearly, the descent direction ∇e is calculated as:

$$\nabla e = [\frac{\partial e}{\partial w_1}, \quad \dots \quad ,\frac{\partial e}{\partial w_n}] = QW - B.$$

In addition, it is easy to show that $\nabla^2 e = \nabla(\nabla e) =$

$$\begin{bmatrix} \frac{\partial^2 e}{\partial w_1^2} & \frac{\partial^2 e}{\partial w_1 \partial w_2} & \dots & \frac{\partial^2 e}{\partial w_1 \partial w_n} \\ \frac{\partial^2 e}{\partial w_2 \partial w_1} & \frac{\partial^2 e}{\partial w_2^2} & \dots & \frac{\partial^2 e}{\partial w_2 \partial w_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 e}{\partial w_n \partial w_1} & \frac{\partial^2 e}{\partial w_n \partial w_2} & \dots & \frac{\partial^2 e}{\partial w_n^2} \end{bmatrix} = Q.$$

In particular, having fixed a descent direction $\nabla e(W^{(t)})$, we can determine the optimal value of the acceleration parameter μ_t that appears in Eq. (3.13), in such a way as to find the point where the function e, restricted to the direction $\nabla e(W^{(t)})$, is minimized. Setting to zero the directional derivative, we get:

$$\frac{d}{d\mu_t} e(W^{(t)} + \mu_t.\Delta W^{(t)}) =$$

$$-(\nabla e(W^{(t)}))^T \nabla e(W^{(t)}) + \mu_t.(\nabla e(W^{(t)}))^T Q \, \nabla e(W^{(t)}) = 0,$$

from which the following expression for μ_t is obtained:

$$\mu_t = \frac{(\nabla e(W^{(t)}))^T \nabla e(W^{(t)})}{(\nabla e(W^{(t)}))^T Q \, \nabla e(W^{(t)})}.$$

After substituting these results in Eq. (3.13) we obtain:

$$W^{(t+1)} = W^{(t)} + \Delta W^{(t)}, \tag{3.15}$$

where

$$\Delta W^{(t)} = -\mu_t.\nabla e(W^{(t)}) + \alpha.\Delta W^{(t-1)}.$$

Notice that, the linear system (3.3) may have no solution. In this case there is no hope to make the error measure close to zero.

Learning algorithm

Step 1: $\alpha > 0$ and $Emax > 0$ are chosen. Then the real vector $W = (w_1, ..., w_n)$ are initialized at random values.
Step 2: Let $t := 0$ where t is number of iterations of the learning algorithm.
Step 3: Take $t := t + 1$. Then the running error E is set to 0.
Step 4: Compute the acceleration parameter μ_t and $e(W)$.
Step 5: The weights vector W is updated by using Eq. (3.15).
Step 6: The training cycle is completed. For $E < Emax$ terminate the training session. If $E > Emax$ then a new training cycle by going back to *Step 3*.

4 Examples

It is probably not an overstatement that linear systems problem arise in almost all practical applications. This section presents two examples of the linear systems to show the behavior and properties of the techniques and also illustrates difference speeds of convergence between the present techniques for the present neural net. For each example, the computed values of the approximate solution are calculated over a number of iterations and also the cost function is plotted.

Example 4.1 *Consider the following linear system of equations*

$$\begin{cases} 4x_1 + 3x_2 + 2x_3 + x_4 = \ \ 4 \\ -x_1 + \ x_2 + 2x_3 + x_4 = -3 \\ 2x_1 + \ x_2 - 2x_3 + x_4 = \ \ 4 \\ \ x_1 + \ x_2 + \ \ x_3 - x_4 = \ \ 2 \end{cases},$$

where $x_i \epsilon R$ $(for\ i = 1, ..., 4)$ with the exact vector solution $(x_1, x_2, x_3, x_4) = (1, 1, -1, -1)$. The training starts by $W = (2, 0, 0, -2)$, $\eta = \frac{1}{15}$ and $\alpha = \frac{1}{15}$. In this example, we apply the proposed techniques to approximate solution of the present system.

Tables 1 and 2 show the approximated solutions over a number of iterations for the steepest descent and quasi-Newton methods, respectively. Fig. 3 shows the difference between the cost functions over the number of iterations. Figs. 4 and 5 show the convergence property of the computed values of the connection weights for the steepest descent and quasi-Newton methods, respectively.

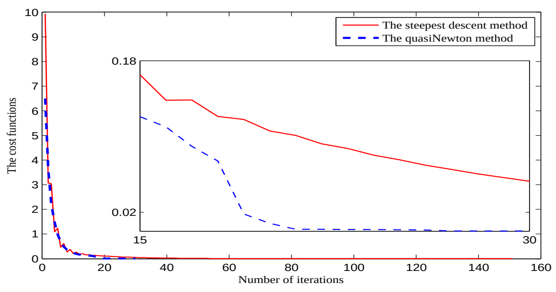

Figure 2: The cost functions obtained from the steepest descent and quasi-Newton methods for Example 4.1 over the number of iterations.

Example 4.2 *Analysis of a processing plant consisting of interconnected reactors*
Consider a chemical processing plant consisting of six interconnected chemical reactors with different mass flow rates of a component of a mixture into and out of the reactors. We are interested in knowing the concentration of the mixture at different reactors. The example here is given in [6]. Application of conservation of mass to all these reactors results in a linear system of equations as follows, consisting of five equations in five unknowns. The solution of the system will be tell us the concentration of the component at each of these reactors. Consider first a reactor with two flows coming in and one flow going out (see figure 5).

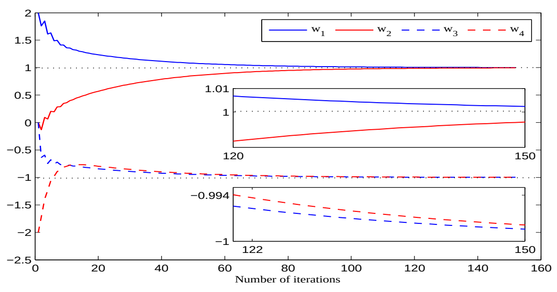

Figure 3: Convergence of the approximated solution obtained from the steepest descent method for Example 4.1.

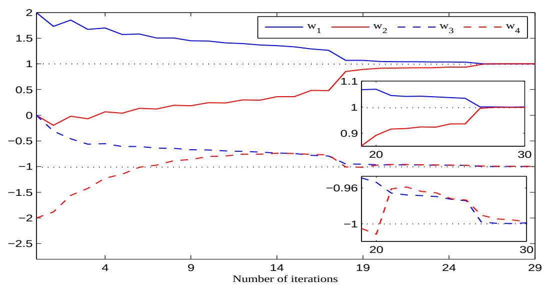

Figure 4: Convergence of the approximated solution obtained from the quasi-Newton method for Example 4.1.

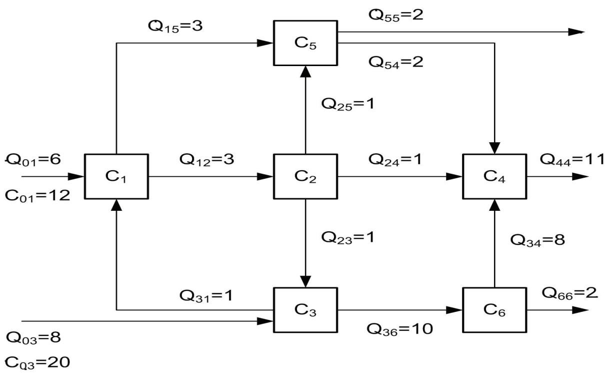

Figure 5: Plant layout consisting of interconnected reactors for Example 4.2.

Table 1: The approximated solutions with error analysis obtained from the steepest descent method for Example 4.1.

t	$W = (w_1, w_2, w_3, w_4)$	e
1	(1.75 -0.13 -0.63 -1.71)	9.9321
2	(1.84 0.09 -0.59 -1.39)	3.0692
3	(1.61 0.06 -0.74 -1.23)	3.0422
4	(1.63 0.20 -0.67 -1.06)	1.0972
$\vdots$	$\vdots$	$\vdots$
150	(1.00 0.99 -0.99 -0.99)	0.000010
151	(1.00 0.99 -0.99 -0.99)	0.000009

Table 2: The approximated solution with error analysis obtained from the quasi-Newton method for Example 4.1.

t	$W = (w_1, w_2, w_3, w_4)$	e
1	(1.72 -0.19 -0.31 -1.88)	6.5000
2	(1.85 -0.01 -0.45 -1.56)	3.6462
3	(1.67 -0.06 -0.56 -1.42)	2.2016
4	(1.69 0.06 -0.55 -1.22)	1.4710
$\vdots$	$\vdots$	$\vdots$
29	(1.00 0.99 -0.99 -0.99)	0.000026
30	(1.00 0.99 -0.99 -0.99)	0.000008

Application of the steady-state conversion of mass for a component in the mixture gives $m_1 + m_2 = m_3$. Noting that $m_i = Q_i.C_i$, where
m_i = mass flow rate the component at the inlet and outlet section i, $i = 1, 2, 3$ in mg/min.
Q_i = volumetric flow rate the section i, $i = 1, 2, 3$ in/m^3/min.
C_i = density or concentration at the section i, $i = 1, 2, 3$ in mg/m^3.
Now we conclude that

$$Q_1C_1 + Q_2C_2 = Q_3C_3. \tag{4.16}$$

For given inlet flow rates and concentrations, the outlet concentration C_3 can be found from above equation, this outlet concentration also represent the homogeneous concentration inside the reactor. Refer now to Fig. 6, where we consider the plant consisting of six reactors. We have the following equations (a derivation of each of these equations is similar to that of Eq. (4.16)):

For reactor 1:

$$6C_1 - C_3 = 72.$$

For reactor 2:

$$3C_1 - 3C_2 = 0.$$

For reactor 3:

$$-C_2 + 11C_3 = 160.$$

For reactor 4:

$$C_2 - 11C_4 + 2C_5 + 8C_6 = 0.$$

For reactor 5:

$$3C_1 + C_2 - 4C_5 = 0.$$

For reactor 6:

$$10C_3 - 10C_6 = 0.$$

Above equations can be written in matrix form as:

$$\begin{bmatrix} 6 & 0 & -1 & 0 & 0 & 0 \\ 3 & -3 & 0 & 0 & 0 & 0 \\ 0 & -1 & 11 & 0 & 0 & 0 \\ 0 & 1 & 0 & -11 & 2 & 8 \\ 3 & 1 & 0 & 0 & -4 & 0 \\ 0 & 0 & 10 & 0 & 0 & -10 \end{bmatrix} \begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_5 \\ C_6 \end{bmatrix} = \begin{bmatrix} 72 \\ 0 \\ 160 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

In this case the training starts by $W = (5, 5, 5, 5)$,

Table 3: The approximated solutions with error analysis obtained from the steepest descent method for Example 4.2

t	$W = (w_1, w_2, w_3, w_4, w_5)$	e
1	(7.38 4.03 5.07 4.90 5.29 17.35)	15.05
2	(8.40 2.19 9.40 16.00 3.911 17.25)	13.94
3	(9.14 2.42 19.53 12.97 5.00 16.09)	7.41
4	(10.09 3.57 16.10 12.92 5.63 13.87)	2.14
$\vdots$	$\vdots$	$\vdots$
165	(14.64 14.64 15.87 15.54 14.64 15.87)	0.001

Table 4: The approximated solutions with error analysis obtained from the steepest descent method for Example 4.2

t	$W = (w_1, w_2, w_3, w_4, w_5)$	e
1	(6.33 4.47 10.52 5.00 5.00 5.00)	71.50
2	(7.78 4.26 10.21 4.96 5.08 8.33)	37.64
3	(8.69 3.95 12.15 6.59 4.95 8.23)	25.14
4	(9.42 3.92 11.62 7.08 5.11 10.25)	19.43
$\vdots$	$\vdots$	$\vdots$
169	(14.64 14.64 15.87 15.54 14.64 15.87)	0.008

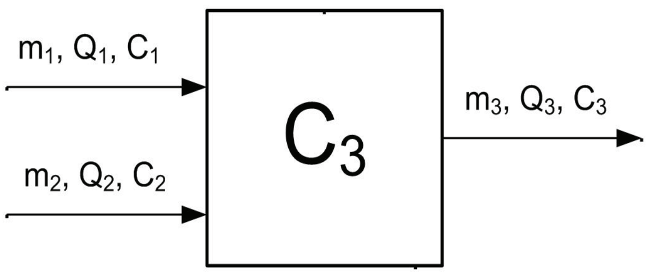

Figure 6: The outlet concentration C_3.

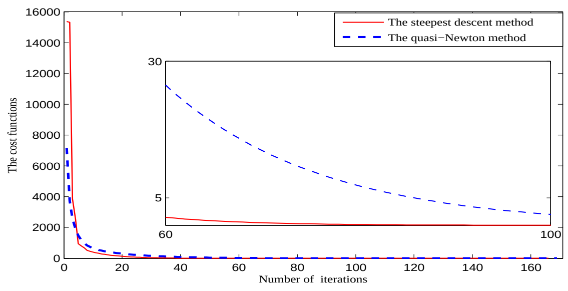

Figure 7: The cost functions obtained from the steepest descent and quasi-Newton methods for Example 4.2 over the number of iterations.

$\eta = \frac{1}{100}$ and $\alpha = \frac{1}{100}$. Similarly, Tables 3 and 4 show the approximated solutions over a number of iterations for the steepest descent and quasi-Newton methods, respectively. Fig. 7 shows the difference between the cost functions over the number of iterations. Figs. 8 and 9 show the convergence property of the computed values of the connection weights for the steepest descent and quasi-Newton methods, respectively.

5 Conclusion

This paper first surveyed the application of neural networks to a system of linear equations and also demonstrated how neural networks have been used to find the unique solution of the present problem. Hence an architecture of feed-forward neural networks has been considered to approximate solution of the linear system. Presented neural net in this study was a numerical method for calculating unknowns in the given system. Then two famous learning techniques namely, the steepest descent and quasi- Newton methods were presented to adjust the connection weights of the neural network. It is clear that to get the best approximating solution of the problem, number of iterations must be chosen large enough. The analyzed examples illustrated the ability and reliability of the techniques. As showed the process of adjustments of the steepest descent and quasi-Newton methods are same. But in the steepest descent technique the connection weight w_j is ad-

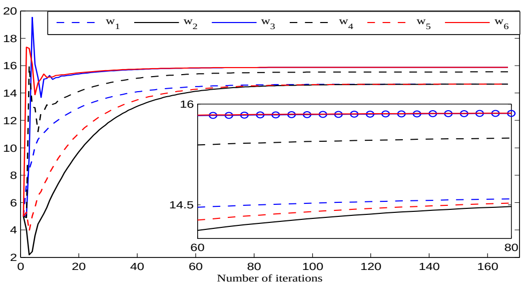

Figure 8: Convergence of the approximated solution obtained from the steepest descent method for Example 4.2.

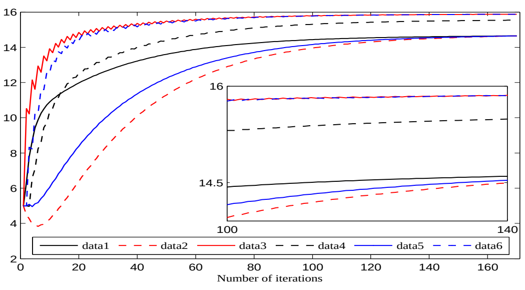

Figure 9: Convergence of the approximated solution obtained from the quasi-Newton method for Example 4.2.

justed for each given learning pattern $\{A_p; b_p\}$. In other words, in solving the linear system of the form $A_{n\times n}X_{n\times 1} = B_{n\times 1}$, in each learning stage the connection weights w_j $(for\ j = 1, ..., n)$ are adjusted n times. However the quasi-Newton technique updates the given wights in each learning stage only one time. Moreover, in the steepest descent method the learning rate η is given by user. This means that the learning rate has same constant value in each learning stage. But a basic problem in this rule is that, by choosing an unsuitable value for η the convergence speed is changed and also the method dont converge to the desired solution. If η catch a small value then the number of iterations go to the high. If the learning rate catch a large value then the calculated solution vacillate environs of the actual solution and never converges to the solution. But in the quasi-Newton method the acceleration parameter μ_t is obtained separate values in in each learning stage. It make that the parameter μ_t takes the best value in each stage. obtained solutions in comparison with exact solutions admit a remarkable accuracy.

Acknowledgment

This paper is based on the Master's project at the Department of Mathematics, Urmia Branch Islamic Azad University of Iran, under graduate research fellowship and fundamental research grant scheme.

References

[1] J. E. Dayhoff, *Neural Network Architectures: An Introduction*, Van Nostrand Reinhold. New York 1990.

[2] R. Fuller, *Neural Fuzzy Systems*, Abo Akademi University 1995.

[3] D. O. Hebb, *The Organization of Behavior*, Wiley. New York 1949.

[4] W. S. McCulloch, W. A. Pitts, *A logical calculus of the ideas imminent in nervous activity*, Bull. Math. Biophys 5 (1943) 115-133.

[5] M. Minsky, S. Papert, *Perceptrons*, MIT Press. Cambridge. Mass. 1969.

[6] N. C. Nash, *Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation-Second Edition*, New York 1990.

[7] A. Quarteroni, R. Sacco, F. Saleri, *Numerical Mathematics*, Springer press. New York 2000.

[8] J.M. Zurada, *Introduction to Artificial Neural Systems*, West Publishing Company. New York 1992.

Ahmad Jafarian was born in 1978 in Tehran, Iran. He received B. Sc (1997) in Applied mathematics and M.Sc. in applied mathematics from Islamic Azad University Lahi-jan Branh to Lahijan, Ferdowsi University of Mashhad respectively. He is a Assistant Prof. in the department of mathematics at Islamic Azad University, Urmia Branch, roumieh, in Iran. His current interest is in artificial intelligence, solving nonlinear problem and fuzzy mathematics.