International Journal of Advance Research in Science and Engineering
Volume No.07, Special Issue No.03, February 2018
www.ijarse.com
IJARSE
ISSN: 2319-8354

Design of Neural Network Model for Linear Problem Solution

Prateeksha Chouksey[1] , Priyanka Lonkar[2], Sampada Chaudhari[3]

[1,2,3]Dept. of Computer Engg., BSCOER, Pune ,(India)

ABSTRACT

Linear programming problem is an optimization problem which helps to find out the optimum value in many optimization problems. It involves lot of application in various domain such as producer consumer problem, image processing, data mining, graph mining. Here we are solving linear programming problem by using neural network. Different training algorithms such as feed forward network, hopfield network and back propagation network are studied, where back propagation algorithm found as most suitable for solving. Back propagation algorithm with two hidden layers is used to train the network. The trained network is then used to solve new linear programming problem which is present into dataset. As linear programming has lots of applications, solving shortest path problem is of great interest of many researchers. As an application we have solved shortest path problem by formulating it into linear programming problem. It is tested for 5 nodes graph as well as 15 nodes graph and in both cases it performs well. Lastly performance of the proposed algorithm is discussed in more detail.

Keywords: Artificial neural network, Back propagation algorithm, Linear programming, Hopfield network, Optimization problem.

I. INTRODUCTION

Linear programming problems arise in a wide variety of scientific and engineering fields including image restoration, parameter estimation, filter design, robot control, etc.; a real-time solution is often desired. Linear programs are problems that can be expressed in canonical form:

$$\text{Maximize } c\ T\ x$$
$$\text{Subject to } Ax \leq b$$
$$\text{And } x \geq 0.$$

Where, x represents the vector of variables (to be determined), c and b are vectors of (known) coefficients, A is a (known) matrix of coefficients, and $(.)^T$ is the matrix transpose. The maximization or minimization function is called the objective function (for example c^Tx). The inequalities constraints are $Ax \leq b$ which specify a convex polytype over which the objective function is to be optimized. In this case, two vectors are comparable when they have the same dimensions. If each entry in the first is less-than or equal-to the corresponding entry in the second then we can say the first vector is less-than or equal-to the second vector. Generally this linear programming problems are solved by simplex method, a univariate search technique etc. To solve linear programming problem by using simplex method manually it will require lot of time. However, traditional numerical methods might not be efficient for digital computers since the computing time required for a solution is greatly dependent on the dimension and the structure of the problem and the complexity of the algorithm used. One promising approach to handle these optimization problems with high dimension and dense structure is to employ artificial-neural-network-based implementation.

An Artificial Neural Network (ANN), usually called neural network (NN), is a mathematical model or computational model that is inspired by the structure and/or functional aspects of biological neural networks. A neural network consists of a group of interconnected artificial neurons , and it processes information using a connectionist approach to computation. In most context an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network during the learning phase. Modern neural networks are non-linear stastical data modeling tools. They are generally used to model complex relationships between inputs and outputs or to find patterns in data. The main benefits of using ANN on linear programming problems are as follows:

i) the ability of training the neurons.

ii) the facility of implementation of circuit in hardware.

iii) the capability of mapping complicated systems while not necessity of knowing the ultimate mathematical models related to them.

But in the past decade very limited work had been done in the field of linear programming using neural network. In 1986, Tank and Hopfield [1] proposed a neural network for solving linear programming problems which was mapped onto a closed-loop circuit. Although the equilibrium point of Tank and Hopfield network may not be a solution of the real problem, this important work has inspired many researchers to investigate other neural networks for solving linear and nonlinear programming problems. Kennedy and Chua [2] extended the Tank and Hopfield network by developing a neural network for solving nonlinear programming problems, by satisfying the Karush–Kuhn–Tucker optimality conditions [9]. The network proposed by Kennedy and Chua contains a penalty parameter which generates approximate solutions only and implementation problems arise when the penalty parameter is large. To avoid the use of penalty parameters, important work has been carried out in recent years. For example, Rodriguez-Vazquez et al. [16] proposed a switched capacitor neural network for solving a class of nonlinear convex programming problems. This network is suitable only for cases in which the optimal solutions lie within the feasible region. Otherwise, the network may have no equilibrium point [30]. Although the model proposed in [31] overcomes the aforementioned drawbacks and is robust for both continuous and discrete-time implementations, but still, the main disadvantage of the network is the requirement to use plenty of rather expensive analog multipliers for variables. Thus, not only the cost of the hardware implementation is very expensive, but also accuracy of solutions is greatly affected. The network of Xia [32] is an improvement over the proposal in [31] in terms of accuracy and implementation cost. The network we discuss here will be both more efficient and less costly than Xia et al.'s [31]. Among these networks back propagation is found to be the most suitable network.

A neural network model is implemented to solve linear programming problem by using back propagation algorithm. Back propagation algorithm is used to train the network. As already we know that there are several applications of linear programming problem. Shortest path problem is one of them. We are solving this shortest path problem by formulating it into linear programming problem and then solve it by using neural network. Shortest path problem is the problem where we have to find the shortest path from source to destination. This shortest path problem can be solved by various algorithms such as dijkstra algorithm, bellman ford algorithm etc. But to solve this shortest path problem by using dijkstra algorithm manually, it will consume more time. Here, we had implemented it online by using neural network approach so that it will take less computation time.So this above methodology is divided into three modules.

The paper contains various sections and it is as follows. The section II contains the design & implementation of the proposed work. Section III gives result analysis. Section IV shows results obtained in every module & finally Section V concludes the work of this study and Section VI points out our future work.

II.DESIGN & IMPLEMENTATION

The above proposed work is implemented in the form of follwing three modules.

2.1. Implementation of Neural Network Model.

2.2. Implementation of Linear Programming Problem.

2.3. Solving Linear Programming Problem by Using Neural Network.

2.1 Implementation of Neural Network Model

Here a neural network model is built by using back propagation algorithm. Back propagation algorithm is used to train the neural network model. This model is specially designed for checking the working of back propagation algorithm on any simple problem. So we will get the idea about the working of an algorithm. From the result we can see that what is the actual output of an example and what is the predictive output which we will get after solving the designed network and the error rate between them. Here we set the threshold value as 10^{-2}. The back propagation process continues until the condition gets satisfy. And when the condition gets satisfy it stops and it will be considered as output.

Back propagation is a common method of training artificial neural networks so as to minimize the objective function. It is a supervised learning rule, and is a generalization of the delta rule. It requires a dataset of many inputs and its desired output , making up the training set. It is most useful for feed-forward networks. The term is an abbreviation for "backward propagation of errors". Back propagation requires that the activation function used by the artificial neurons(or "nodes") be differentiable. The back propagation learning algorithm can be divided into two phases: propagation and weight update.

Phase 1: Propagation

Each propagation consists of following steps:

1. Forward propagation of a training pattern's input through the neural network in order to generate the propagation's output activations.
2. Backward propagation of the propagation's output activations through the neural network using the training pattern's target in order to generate the deltas of all output and hidden neurons.

Phase 2: Weight update

For each weight-synapse follows the following steps:

1. Multiply its output delta and input activation to get the gradient of the weight.
2. Bring the weight in the opposite direction of the gradient by subtracting a ratio of it from the weight.

This ratio influences the speed and quality of learning; it is called the learning rate. The sign of the gradient of a weight indicates where the error is increasing; this is why the weight must be updated in the opposite direction.

Repeat phase 1 and 2 until the performance of the network is satisfactory.

There are two modes of learning to choose from: One is on-line (incremental) learning and the other is batch learning. In on-line (incremental) learning, each propagation is followed immediately by a weight update. In batch learning, much propagation occur before weight updating occurs. Batch learning requires more memory capacity, but on-line learning requires more updates. As the algorithm's name implies, the errors propagate backwards from the output nodes to the inner nodes. Technically speaking, back propagation calculates the gradient of the error of the network regarding the network's modifiable weights. This gradient is almost always used in a simple stochastic gradient descent algorithm to find weights that minimize the error. Often the term "back propagation" is used in a more general sense, to refer to the entire procedure encompassing both the calculation of the gradient and its use in stochastic gradient descent. Back propagation usually allows quick convergence on satisfactory local minima for error in the kind of networks to which it is suited.

2.2 Implementation of Linear Programming Problem

In this step, the linear programming problem is solved by using simple procedure such as simplex method. The general linear programming problem formulation is as follows

$$\text{Minimize } c^T x$$
$$\text{Subject to } Ax \le b$$
$$\text{and } x \ge 0$$

The study of duality is very important in linear programming. Tha knowledge of duality allows one to develop increased insight into linear programming solution interpretation. Also, when solving the dual of any problem, one simultaneously solves the primal. Thus, duality is an alternative way of solving linear programming problems. However, given today's computer capabilities, this is an infrequently used aspect of duality. Therefore, we concentrate on the study of duality as a means of gaining insight into the LP solution. We will also discuss the ways that primal decision variables place constraints upon the resource valuation information. The Primal problem can be written as:

$$\text{Max} \sum C_j X_j$$
$$\text{s.t } \sum a_{ij} Xj \le b_i \text{ for all i}$$
$$X_j \ge 0 \text{ for all j}$$

Associated with this primal problem is a dual resource valuation problem. The dual of the above problem is

$$\text{Min} \sum U_i b_i$$
$$\text{s.t} \sum U_i a_{ij} \ge c_j \text{ for all j}$$
$$U_i \ge 0 \text{ for all i}$$

where U_i are the dual variables.

If the primal problem has n variables and m resource constraints, the dual problem will have m variables and n constraints. There is a one-to-one correspondence between the primal constraints and the dual variables; i.e., U1 is associated with the first primal constraint, U2 with the second primal constraint, etc. As we demonstrate later, dual variables (Ui) can be interpreted as the marginal value of each constraint's resources. These dual variables are usually called shadow prices and indicate the imputed value of each resource. A one-to-one correspondence also exists between the primal variables and the dual constraints; X1 is associated with the first dual constraint and X2 is associated with the second dual constraint.

This linear programming problem is solved in MATLAB. It runs for some problems to prepare test cases which will be used in training stage in final module. The data sets of 110 problems are created which will be used in training phase of a network in final module.

2.3 Solving Linear Programming Problem by Using Neural Network

This module is designed to solve linear programming problem by using neural network. Working of the model is as follows:

Consider a linear programming problem in the following standard form:

Find x that

Maximizes: $b^T x$

Subject to the constraints: $A x \le c, \quad x \ge 0$ (1)

Where x and $b \in R^n$, $A \in R^{m \times n}$, and $c \in R^m$.

The dual problem of (1) is:

Find y that

Minimizes: $c^T y$

Subject to the constraints: $A^T y \ge b, \quad y \ge 0$……………………………….(2)

Mathematically, the outputs of the above primal and dual neurons can be described by the following nonlinear dynamical system:

$dx/dt = b - A^T (y + k (dy/dt))$, $x \ge 0$……………………(3)

$dy/dt = -c + A (x + k (dx/dt))$, $y \ge 0$…………………….(4)

It can be seen that (3) and (4) are equivalent to a system of second order differential equations.

The Euler method is used to solve differential equations (3) and (4) which will be solved in neural network processing.

To solve this linear programming problem we have to train the network model by using back propagation algorithm. Training stage of a network model is as follows:

Algorithm:

Input:

1. Pair of Linear Programming Problems with Solution
2. Minimum error value

Output:

```
Training Stage:
1. Initialize the weights in the network (often randomly)
2.  Do
        For each example data (e) in the training set
            i. O= neural-net-output (network, e);
            ii. T = output for e
            iii. Calculate error (T - O) at the output units
            iv. Compute delta_wi for all weights from hidden
                layer to output layer ;
            v. backward pass Compute delta_wi for all  weights
               from  input layer to hidden layer;
            vi. backward pass continued
            vii. Update the weights in the network until dataset
                 classified correctly or stopping criterion satisfied
3.   Return the network.
```

Testing Stage:

Simulate the network with new input (testing) problem.

In the processing T is calculated by using Euler's method. By using various test cases neural network will be train and that network will be used to solve linear programming problem. Expected output is a solution for the problem.

Here, 111 linear programming problems were solved by using built in mat lab function for linear programming out of which 109 problems are taken for training and 2 are taken for testing. Corresponding outputs were also added in .mat file. Back- propagation algorithm, mentioned above was executed on training input. While training the single problem on to the designed back propagation neural network model. We have to set the stoppage criteria otherwise the process continues and we will never get a output. The criteria is as follows

i. Set the threshold value as 10^{-2}

ii. It will back propagate upto 3000 iterations only.

iii. Validation checks upto 6 only.

If out of these three condition atleast one condition is satisfied. Then it will stop the process. This whole process runs for five times i.e the process runs for five times to get accuracy. The average of that is considered as final output.

To further extend, the shortest path problem is solved by using linear programming problem. Shortest path problem is defined as finding the shortest path from source node to destination node. There are many paths to reach from source node to destination node. Among that several paths we have to find the shortest path from source node to destination node. To solve this shortest path using linear programming problem we have to convert it into linear programming problem.

Here we assume the network is directed and connected. Let E the node arc incidence matrix of the network, and c is the cost, or distance of arcs. Given a source node s and for any sink node t, we have the following linear programming formulation for the shortest path from s to t

$$d_t = \min cx^{(t)}$$
$$\text{s. t } Ex^{(t)} = e_{st} \quad \text{.............} \quad (1)$$
$$x^{(t)} \geq 0$$

where

$$e_{st}=[0,\ldots -1\ldots 1\ldots .0]$$

and -1 appears at position s and 1 appears at position t.

The dual of above linear programming is

$$d_t = \max e_{st}\,\pi$$
$$E'\pi \leq c.$$

It is interesting to notice that the dual feasible regions are identical for all t. Without loss of generality, we can also set π_s =0. Hence, the dual program can be rewritten as

$$d_t = \max \pi_t$$
$$-\pi i + \pi j \leq cij$$

It is easy to notice that if there exists an optimal solution, then for all i $\pi_i = d_i$

That is, d represents the shortest distance from each node to source node s.

Optimality Condition I. For any sink nod t, the optimal solution of the primal linear program SP (t) and the optimal solution of its dual linear program satisfies the following three conditions. Vice versa.

(Primal Feasibility) $Ex=e_{st} \quad x\geq 0$

(Dual Feasibility) $-d_i + d_j \leq c_{ij}$.............(2)

(Complementary Slackness) $x^{(t)}_{ij} \quad (c_{ij} + d_i + d_j) =0$……………..(3)

Optimality Condition II. The dual optimal solutions must satisfies

$$d_j = \min_{(i,j) \in AI\{j\}} (d_i + c_{ij}) \text{ …………}(4)$$

The reverse is also true if network contains no directed cycle with length zero.

The above equations (4) are also referred as Bellman's equations.

Proposition 1: The Optimality conditions I and II are indeed defines the primal and dual optimal solutions, and they are equivalent.

Shortest Path on Acyclic Graph:

We know that a unique feature for an acyalic graph is that there exists a topological order. Without loss of generality, we assume now the nodes are labeled in a topological order for a given network. Then equation (5.4) becomes a recursive equation

$$d_j = \min_{(i,j) \in AI\{j\}} (d_i + c_{ij})$$

Therefore, all shortest distances can be computed sequentially according to the topological order. The algorithm can be described as

```
S = 1; $d_s$ =0
for j=1:n
        $d_j = \min_{(i,j) \in AI\{j\}, i<j} (d_i + c_{ij})$
```

end_for

We present the Dijkstra's algorithm for finding the shortest path on networks with all arc costs or distances non-negative. The algorithm is very simply and runs in O (n ^{2})

```
Initialization   $d_s = 0$; $d_s j = c_{sf}$ j€ A{s};
                 $d_j = \infty$ , j € N\ A{s} ;
                 U = N \ {s};
while    |U| > 1
         $d_i = \min_{j€ U} d_j$ ;
         U: = U\{s};
         for   j € A(i)
              $d_j = \min(d_j , d_i + c_{ij})$;
         end_for
end_while
```

Theorem c. The Dijkstra's algorithm correctly finds the shortest path on networks with non-negative arc costs or distance.

Proof: We use induction on iteration t to show that

1) $\max_{i \,€\, N\backslash U} d_i \leq \min_{j€ U} d_f$

2) $- d_i + dj \leq c_{ij}$,

(i, j) € $A\backslash A_U$

where

A_U={(i,j)| i,j € U}

Clearly, at first iteration, both 1) and 2) are true. Suppose at the beginning of iteration t both 1) and 2) are true. Let

$d_{i*} = \min_{j \,€\, U} d_j$

We use superscript "+" to denote the updated data at the end of iteration t.

For

j € U\{i*} and $d_j^+ < d_j$

$d_j^+ = d_{i*} + c_{ij} \geq d_{i*} \geq \min_{i \,€\, N \backslash U} d_i$,

and for j € U\{i*} and $d_j^+ = d_j$ it is obvious that 1) is true due to he rule to choose i*.

To show 2) is true at the end of iteration *t*, we discuss several cases.

a) i € {i*}ᵕ (N\U) , j € U \ {i*},

$-d_i + d_j^+ \leq -d_i + d_j \leq c_{ij}$.

b) j € {i*}ᵕ (N\U), i€ U\{i*},

and $d_i^+ < d_i$,

then, because of the induction hypothesis that 1) is true,

$d_j \leq d_i$,

$-d_i^+ + d_j = -(d_{i*} + c_{i*i}) + d_j \leq -(d_{i*} + c_{i*i}) + d_{i*} \leq 0 \leq c_{ij}$.

After the conversion is done into linear programming problem. Then we can solve this problem by using neural network i.e by back propagation algorithm. For example: To solve the below shortest path problem, first it has to be formulated into linear programming problem. Then that formulation is solved by using neural network model to get the optimal solution of a given graph.

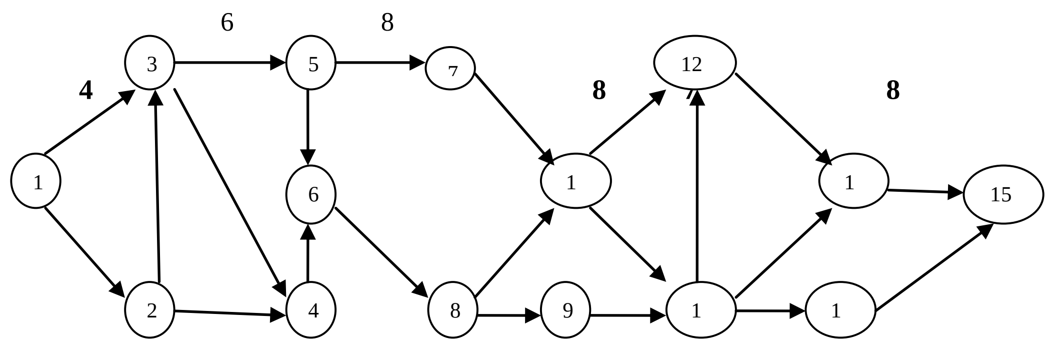


Fig. 1. Shortest path graph

Here we wish to find the shortest path from node 1 to node 15. Dijkstra algorithm is used to find shortest path from node 1 to 15. The value of shortest path is found to be 28.

III. RESULT ANALYSIS

The simulation results are generated by neural network toolbox of mat lab.

3.1. For original back – propagation algorithm

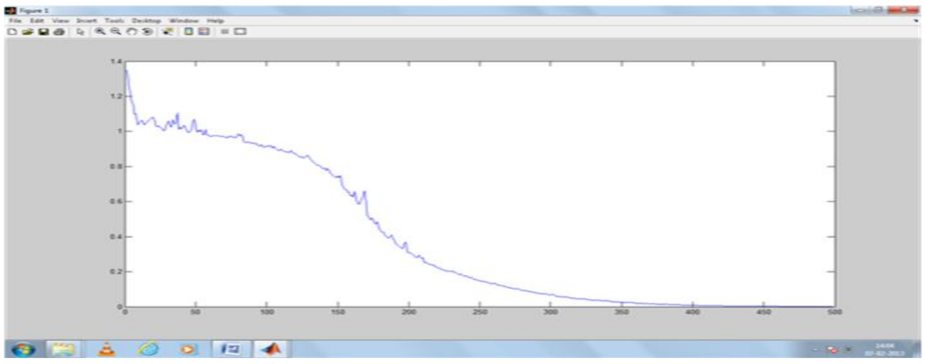


Fig.2. Error rate

Output

```
converged at iterations: 499
state after 499 iterations
act_pred_err =  1.0000    0.9996     0.0004
                     0        0.0008   -0.0008
                     0        0.0004   -0.0004
                1.0000    0.9998    0.0002
```

Analysis

From the above graph we can say that the error rate goes on decreasing as the iterations goes on increasing. In the output we can see the actual output of the problem and the predictive output which is the output after solving the problem using designed back propagation neural network model.

3.2. For original linear programming problem using simplex method

- dsimplex ('max',[1 6],[1 8;7 9],[5 8])

```
ans =

      1
      4
```

- dsimplex('min',[1 6],[1 8;7 9],[5 8])

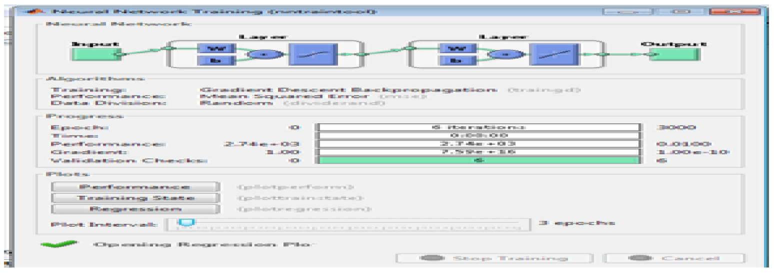
```
ans =

     2
     1
```

Analysis

From the above output we can see that the output of simple simplex method. This is used to create a dataset which will be useful for training the designed three layer neural network structure.

3.3. For solving linear programming problem using back propagation algorithm

- **Output screen**



```
Neural Network Training (nntraintool)
Neural Network
Algorithms
Training:        Gradient Descent Backpropagation (traingd)
Performance:     Mean Squared Error (mse)
Data Division:   Random (dividerand)
Progress
Epoch:                  0        6 iterations      3000
Time:                            0:00:00
Performance:     2.74e+03        2.74e+03          0.0100
Gradient:            1.00        7.59e+16          1.00e-10
Validation Checks:      0        6                 6
Plots
Plot Interval:                                     3 epochs
Opening Regression Plo
```

Input
Layer
W
b
Layer
W
b
Output
Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)
Stop Training
Cancel

Fig. 3. Output Screen

Analysis

From the above figure we can see the structure of neural network model. It is a three layer structure neural network model. As well the number of iterations, performance, gradient and validation checks required to get an optimal solution of a linear programming problem can be seen.

- **Performance**

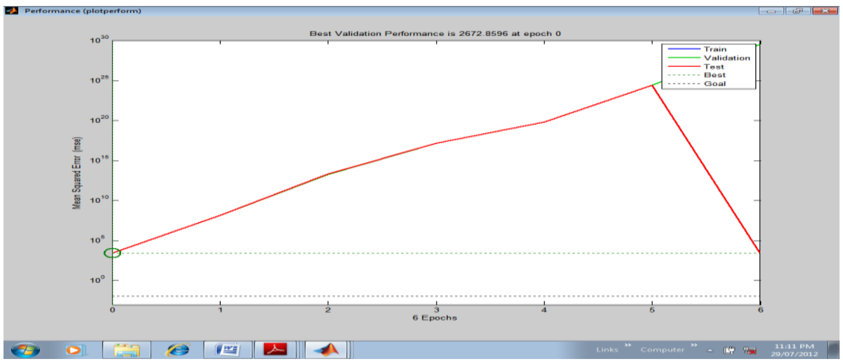


Fig. 4. Performance plot

Analysis

From the above figure we can see the performance of validation, training and testing. The green line indicates the validation performance, blue and red line indicates testing and training performance respectively which is decreasing as number of iteration increases.

- **Gradient**

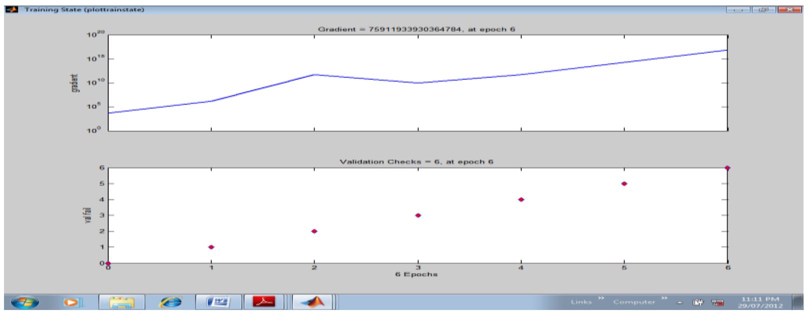
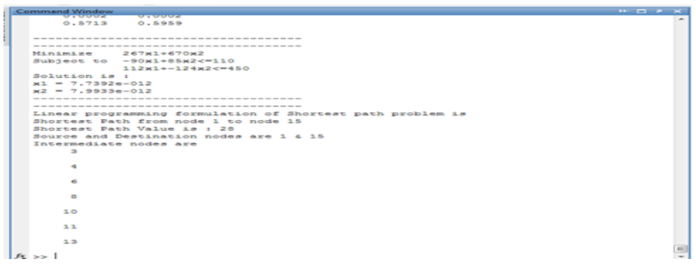


Fig.5. Gradient plot

Analysis

From the figure we can see the performance of gradient and validation checks. It shows that the error rate goes on decreasing as the number of training problems are increasing. As number of problems increases for training the error rate goes on decreasing and we will get accuracy in finding out the optimal solution of the new problems under testing stage.

- **Output of LPP and Shortest path problem**



```
    0.5713     0.5959

------------------------------------
------------------------------------
Minimize      267x1+670x2
Subject to    -90x1+85x2<=110
              112x1+-124x2<=450
Solution is :
x1 = 7.7392e-012
x2 = 7.9933e-012
------------------------------------
------------------------------------
Linear programming formulation of Shortest path problem is
Shortest Path from node 1 to node 15
Shortest Path Value is : 28
Source and Destination nodes are 1 & 15
Intermediate nodes are
     3

     4

     6

     8

    10

    11

    13

>> 
```

Fig.6. Output of LPP and Shortest path problem

Analysis

From the above we can see the output values of x1 & x2 of linear programming problem and the output of shortest path problem from node 1 to 15 is 28 and the intermediate nodes are 3,4,6,8,10,11,13.

IV.CONCLUSION

Linear programming problem is one of the promising problems in any optimization problem, which involves lot of application in various domain such as producer consumer problem, image processing, data mining, graph mining. Various methods are proposed by researchers. Simplex method is one of them. At the same time neural network is found to be efficient evaluating tool in various domains. Due to training provided to the input it is easy to find the pattern and based on the pattern output can be found on test problem. Neural network has its own capability of training the network like feed forward, back propagation, hopfield network. We use back propagation algorithm to solve LP problem. To test the performance we have created our own dataset which is solved by simplex method. Dataset is stored in .mat file where 110 problems were taken for training. Error rate and convergence is shown and discussed here. To further continue we have solved Shortest path problem by using linear programming which gives optimal solution for a given graph. The result for given graph is also shown above.

V.FUTURE SCOPE

Future work involves solving variety of various application problems by implemented neural network model. Similarly, the quadratic problem can be solved by using this network.

REFERENCES

[1] Tank, D.W. and Hopfield, J. 1986, *"Simple neural optimization networks: An A/D converter, signal decision circuit, and a linear programming circuit", IEEE Transactions on Circuits and Systems, 33(5),533–541*.

[2] Kennedy, M.P. and Chua, L.O. 1988*, "Neural networks for nonlinear programming", IEEE Transactions on Circuits and Systems, 35(5), 554– 562.*

[3] Hasan Ghasabi-Oskoei a, Nezam Mahdavi-Amiri,2006, *"An efficient simplified neural network for solving linear and quadratic programming problems", Applied Mathematics & computation Journal, Elsevier, pp 452–464.*

[4] L.R. Arvind Babu and B. Palaniappan, *"Artificial Neural Network Based Hybrid Algorithmic Structure for Solving Linear Programming Problems", International Journal of Computer and Electrical Engineering, Vol. 2, No. 4, August, 2010 , pp 1793-8163.*

[5] G.M. Nasira, S. Ashok kumar, T.S.S. Balaji, *"Neural Network Implementation for Integer Linear Programming Problem", 2010 International Journal of Computer Applications (0975 - 8887) Volume 1 – No. 18.*

[6] Mohsen Alipour, "A Novel Recurrent Neural Network Model For Solving Nonlinear Programming

Problems With General Constraints", Australian Journal of Basic and Applied Sciences, 5(10): 814-823, 2011.

[7] S. Paulraj, C. Chellappan, T. R. Natesan, 2006, "*A heuristic approach for identification of redundant constraints in linear programming models". International Journal of Computer Mathematics, 83:8, 675 –683.*

[8] Neeraj Sahu, Avanish Kumar, *"Solution of the Linear Programming Problems based on Neural Network Approach", IJCA, Volume 9– No.10, November 2010.*

[9] Maa, C.-Y. and Shanblatt, M.A. 1992, *"Linear and quadratic programming neural network analysis", IEEE Transactions on Neural Networks, 3(4), 580–594.*

[10] Alaeddin Malek and Maryam Yashtini, *"A Neural Network Model for Solving Nonlinear Optimization Problems with Real-Time Applications", W. Yu, H. He, and N. Zhang (Eds.): ISNN 2009, Part III, LNCS 5553, pp. 98–108, 2009. © Springer-Verlag Berlin Heidelberg 2009.*

[11] Xingbao Gao and Li-Zhi Liao, *"A New One-Layer Neural Network for Linear and Quadratic Programming", IEEE transactions on neural networks, vol. 21, no. 6, June 2010.*

[12] Ue - Pyng Wen , Kuen - Ming Lan and Hsu - Shih Shi,2009, *"A review of Hopfield neural networks for solving mathematical programming problems", European Journal of Operational Research, Elsevier, pp 675- 687.*

[13] Hong - Xing Li and Xu Li Da , 2000 , *"A neural network representation of linear programming", European Journal of Operational Research, Elsevier, pp 224-234.*

[14] Xiaolin Hu,2009 , *"Applications of the general projection neural network in solving extended linear-quadratic programming problems with linear constraints", Neurocomputing , Elsevier , pp 1131 -1137.*

[15] J. M. Ortega and W. C. Rheinboldt, "*Iterative Solution of Nonlinear Equation in Several Variables", New York: Academic, 1970.*

[16] A. Rodríguez-Vázquez, R. Domínguez-Castro, J. L. Huertas, and E. Sánchez-Sinencio*, "Nonlinear switched-capacitor 'neural networks' for optimization problems," IEEE Trans. Circuits Syst., vol. 37, pp. 384–397, Mar. 1990.*

[17] J.-J. E. Slotine and W. Li, *"Applied Nonlinear Control", Englewood Cliffs, NJ: Prentice-Hall, 1991.*

[18] M. V. Solodov and P. Tseng*, "Modified projection-type methods for monotone variational inequalities", SIAM J. Control Optim., vol. 34, no. 5, pp. 1814–830, 1996.*

[19] Q. Tao, J. Cao, and D. Sun, *"A simple and high performance neural network for quadratic programming problems," Appl. Math. Comput., vol. 124, no. 2, pp. 251–260, 2001.*

[20] P. Tseng, *"A modified forward-backward splitting method for maximal monotone mappings," SIAM J. Control Optim., vol. 38, no. 2, pp. 431–446, 2000*.

[21] J. Wang*, "Primal and dual assignment network," IEEE Trans. Neural Netw., vol. 8, no. 3, pp. 784–790, May 1997.*

[22] M. Avriel, *"Nonlinear Programming: Analysis and Methods", Englewood Cliffs, NJ: Prentice-Hall, 1976.*

[23] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty, *"Nonlinear Programming- Theory and Algorithms", 2nd ed. New York: Wiley,1993.*

[24] A. Bouzerdorm and T. R. Pattison, *"Neural network for quadratic optimization with bound constraints," IEEE Trans. Neural Netw.,vol. 4, no. 2, pp. 293–304, Mar. 1993.*

[25] M. P. Glazos, S. Hui, and S. H.Z´ ak, *"Sliding modes in solving convex programming problems," SIAM J. Control Optim., vol. 36, no. 2, pp. 680–697, 1998.*

[26] Q. M. Han, L.-Z. Liao, H. D. Qi, and L. Q. Qi, *"Stability analysis of gradient-based neural networks for optimization problem," J. Global Optima., vol. 19, no. 1, pp. 363–381, 2001.*

[27] Y. S. Xia, *"A new neural network for solving linear programming an quadratic programming problems," IEEE Trans. Neural Netw., vol. 7, no. 6, pp. 1544–1547, Nov. 1996.*

[28] Y. S. Xia*, "An extended projection neural network for constrained optimization", Neural Comput., vol. 16, no. 4, pp. 863–883, 2004.*

[29] Y. S. Xia and G. Feng, *"A new neural network for solving nonlinear projected equations," Neural Netw., vol. 20, no. 5, pp. 577–589, 2007.*

[30] S.H. Zak, V. Upatising, S. Hui, *"Solving linear programming problems with neural networks: a comparative study", IEEE Transactions on Neural Networks 6 (1) (1995) 96–104.*

[31] X. Wu, Y. Xia, J. Li, W. Chen, *"A high performance neural network for solving linear and quadratic programming problems", IEEE Transactions on Neural Networks 7 (3) (1996) 643– 651.*

[32] Y. Xia, *"A new neural network for solving linear programming problems and its application", IEEE Transactions on Neural Networks 7 (2) (1996) 525–529.*

[33] Y. S. Xia and J. Wang, *"A dual neural network for kinematic control of redundant robot manipulators," IEEE Trans. Syst. Man Cybern. B, Cybern., vol. 31, no. 1, pp. 147–151, Feb. 2001.*

[34] Y. S. Xia and J.Wang, *"A general projection neural network for solving monotone variational inequalities and related optimization problems," IEEE Trans. Neural Netw., vol. 15, no. 2, pp. 318–328, Mar. 2004.*

[35] Y. S. Xia and J. Wang, *"A recurrent neural network for nonlinear convex optimization subject to nonlinear inequality constraints," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 51, no. 7, pp. 1385–1394, Jul. 2004.*

[36] Y. S. Xia and J. Wang, *"Solving variational inequality problems with linear constraints based on a novel neural network," in Lecture Notes in Computer Science. Berlin, Germany: Springer-Verlag, 2007, vol. 4493, pp. 95–104.*

[37] Y. Zhang and J. Wang, *"A dual neural network for convex quadratic problem* subject to linear equality *and inequality constraints," Phys. Lett. A, vol. 298, no. 4, pp. 271–278, 2002.*