ARTIFICIAL NEURAL NETWORK IMPLEMENTATION FOR SOLVING LINEAR PROGRAMMING MODELS

Dr. ABDUL KAREEM F. HASSAN FATH ALLAH FADHIL KHALAF
Mechanical Engineering Department
College of Engineering
University of Basrah

Abstract

This work does not use the classical methods (simplex method, Branch and Bound techniques) which were normally used for solving Linear programming models. The proposed algorithm was considered for implementation with Artificial Neural Network (ANN) using MatLab tool box. It was found that implementation of the neural network will provide comprehensive results when applied with any linear programming models. Besides Artificial Neural Networks are artificial intelligence methods for modeling complex target functions, and are considered to be among the most effective learning methods currently known. Implementation in solving linear programming models became very interesting, as ANNs became appropriate solution where a huge data (number of variables and constraints) is considered. In this work, general model of ANN specified for solving the problem of linear programming will be shown and discussed. The results show a great improvement in prediction of results with a minimum percentage error.

Key words: linear programming, neural network.

1. INTRODUCTION

A linear programming (LP) problem is a mathematical program in which the objective function is linear in the unknowns and the constraints consist of linear equalities. LP has long proved its merit as a significant model for production planning and scheduling, numerous allocation, operations research, economic problems, and so forth[1].The simplex method developed by Dantzig, is still the most widely used numerical algorithm. Although the simplex method is efficient and elegant, but the modern numerical algorithms are very efficient and useful to solve LP problem. Two new neural network models for solving the maximum flow problem are presented by S. Effati and M. Ranjbar[2]. The maximum flow problem in networks is formulated as a special type of linear programming problem and it is solved by appropriately defined neural networks. Optimization of the machining parameters for each pass were studied by SezgiÖzen and G. MiracBayhan[3]. The developed algorithm is based on minimum unit cost criterion and the objective of the problem is to minimize unit production cost without violating any technological, economic and organizational constraints. A Hopfield-type dynamical network which employs a penalty function approach is proposed for solving the problem formulated by mixed integer linear programming. A new technique to control the voltage and reactive power in power systems based on Artificial Neural Network (ANN) was prepared by M. Joorabianand R. Hooshmand[4]. Feed-forward ANN with Back-Propagation training algorithm is used and the training data is obtained by solving several abnormal conditions using Linear Programming (LP). J. AbdulSamath et al. [5] introduce a neural network algorithm to study the singular system of a linear electrical circuit for time invariant and time varying cases. The discrete solutions obtained using neural network, are compared with Runge-Kutta method and exact solutions of the electrical circuit problem and are found to be very accurate. A new design methodology for the general projection neural network(GPNN) is proposed by Xiaolin Hu and Jun Wang[6]. Different types of constraints, approaches for reducing the number of neurons of the GPNN are discussed, which results in two specific GPNNs. Moreover, some distinct properties of the resulting GPNNs are also explored based on their particular structures. An illustration of linear programming in a tutorial style and its use evolved are outlined in three areas: generation scheduling, loss minimization through allocation of reactive power supply, and planning of capital investments in generation equipment[7]. The applications include not only linear programming but also its extensions to integer and quadratic programming and to the use of Benders and Danuig-Wolk decomposition techniques.

2. PROBLEM FORMULATION

Mathematically, linear programming deals with nonnegative solutions to linear equation systems. The classical linear programming problem is to find a vector (x_1, x_2, …, x_n) which maximizes the linear form (i.e. the objective function)[1].

$$Max. or\ Min. \quad Z = \sum_{j=1}^{n} c_j x_j$$

$$subject\ to:$$

$$\sum_{j=1}^{n} a_{ij}\,.x_j \le b_i \quad i = 1,2,\dots\dots,m$$

$$x_j \le 0, \ j = 1,2,\dots\dots,n$$

In matrix table

x_1	x_2	RHS
a_{11}	a_{12}	b_1
a_{21}	a_{22}	b_2
…….	…….	…….
a_{n1}	a_{n2}	b_n

3. Theoretical Background of artificial Neural Network

Neural networks can be thought of as "black box" devices that accept inputs and produces outputs. Figure(1) shows a typical neural network structure consisting of three layers:

Input Layer: A layer of neurons that receives information from external sources, and passes this information to the network for processing. These may be either sensory inputs or signals from other systems outside the one being modeled.

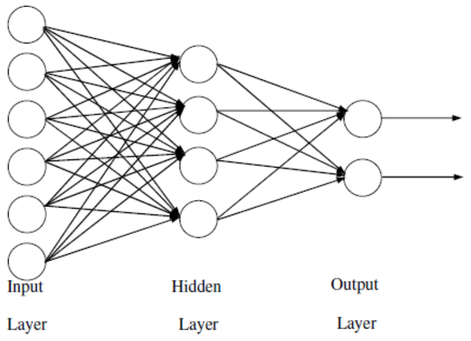


Fig. (1) Structure of Neural Network

Hidden Layer: A layer of neurons that receives information from the input layer and processes them in a hidden way. It has no direct connections to the outside world (inputs or outputs). All connections from the hidden layer are to other layers within the system.

Output Layer: A layer of neurons that receives processed information and sends output signals out of the system.

Bias: Acts on a neuron like an offset. The function of the bias is to provide a threshold for the activation of neurons. The bias input is connected to each of the hidden and output neurons in a network.

The number of input neurons corresponds to the number of input variables into the neural network, and the number of output neurons is the same as the number of desired output variables. The number of neurons in the hidden layer(s) depends on the application of the network.

As inputs enter the input layer from an external source, the input layer becomes "activated" and emits signals to its neighbors (hidden layer) without any modification. Neurons in the input layer act as distribution nodes and transfer input signals to neurons in the hidden layer. The neighbors receive excitation from the input layer, and in turn emit an output to their neighbors (second hidden layer or output layer). Each input connection is associated to a quantity, called “a weight factor” or “aconnection strength”. The strength of a connection between two neurons determines the relative effect that one neuron can have on another. The weight is positive if the associated connection is excitatory, and negative if the connection is inhibitory.

3.1 Elements of Neural Network

Artificial neurons as information processing devices were first proposed more than sixty years ago. Following this early work, the pattern recognition capabilities of preceptrons, in which the neurons are arranged in layers, were investigated both theoretically and experimentally throughout the 1950s by Rosenblatt and others. As shown in Figure(2), a neuron computes a weighted summation of its n inputs, the results of which is then threshold to give a binary output b_i.

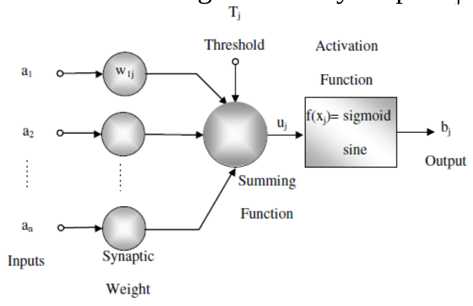


Fig. (2) Single Node Anatomy

A. Inputs and Outputs

Inputs are represented by a_1, a_2, …,a_n, and the output by b_j. Just as there are many inputs to a neuron, there should be many input signals to the PE. The PE manipulates these inputs to give a single output signal.

B. Weighting Factors

The values w_{1j}, w_{2j}, …, and w_{nj} are weight factors associated with each input to the node. This is something like the varying synaptic strengths of biological neurons. Weights are adaptive coefficients within the network that determine the intensity of the input signal. Every input (a_1, a_2, …, a_n) is multiplied by its corresponding weight factor (w_{1j}, w_{2j}, …, w_{nj}), and the node uses this weighted input (w_{1j} a_1, w_{2j} a_2, …, w_{nj} a_n) to perform further calculations. If the weight factor is positive, then (w_{ij} a_i) tends to excites the node. If the weight factor is negative, then (w_{ij} a_i) inhibits the node[8].

In the initial setup of a neural network, weight factors may be chosen according to a specified statistical distribution. Then these weight factors are adjusted in the development of the network or “learning” process.

C. Internal Threshold

The other input to the node is the node’s internal threshold, Tj. This is a randomly chosen value that governs the “activation” or total input of the node through the following equation.

$$\text{Total Activation} = u_j = \sum_{i=1}^{n} \left(w_{ij} a_i\right) - T_j$$

The total activation depends on the magnitude of the internal threshold T_j. If T_j is large or positive, the node has a high internal threshold, thus inhibiting node-firing. If T_j is zero or negative, the node has a low internal threshold, which excites node-firing[9]. If no internal threshold is specified, a zero value is assumed.

D. Transfer Functions

The node’s output is determined by using a mathematical operation on the total activation of the node. This operation is called a transfer function. The transfer function can transform the node’s activation in a linear or nonlinear manner [48]. Figure (3) shows several types of commonly used transfer function.

3.2 Learning or Training of Neural Networks

The property that is of primary significance for a neural network is the ability of the network to learn from its environment, and to improve its performance through learning. The improvement in performance takes place over time in accordance with some prescribed measure. A neural network learns about its environment through an interactive process of adjustments applied to its synaptic weights and bias levels. Ideally, the network becomes more knowledgeable about its environment after each iteration of the learning process.

Therefore, Learning is defined as a process by which the free parameters of a neural network are adapted through a process of simulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place [50].

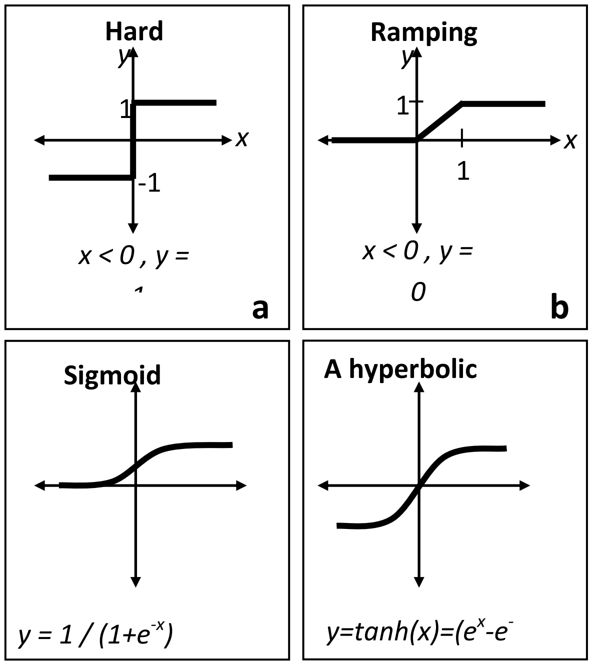


Figure (3) Sample Transfer Functions

This definition of the learning process implies the following sequence of events:

1. The neural network is simulated by an environment.
2. The neural network undergoes changes in its free parameters as a result of this simulation.
3. The neural network responds in a new way to the environment because of the changes that have occurred in its internal structure.

A prescribed set of well-defined rules for the solution of a learning problem is called a learning algorithm.

Supervised Learning: Supervised learning requires an external teacher to control the learning and incorporates global information. The teacher may be a training set of data or an observer who grades the performance. Examples of supervised learning algorithms are the least-mean-square (LMS) algorithm and its generalization, known as the back propagation algorithm.

Unsupervised Learning: It is sometimes called self-supervised learning. Here, networks use no external influences to adjust their weights. Instead there is an internal monitoring of performance. The network looks for regularities or trends in the input signals, and makes adaptations according to the function of the network. Even without being told whether it’s right or wrong, the network still must have some information about how to organize itself.

Reinforcement Learning: This type of learning may be considered as an intermediate form of the above two types of learning. Here the learning machine does some action on the environment and gets a feedback response from the environment. The learning system grades its action good (rewarding) or bad (punishable) based on the environmental response and accordingly adjusts its parameters. Generally, parameter adjustment is continued until an equilibrium state occurs, following which there will be no more changes in its parameters. The self organizing neural learning may be categorized under this type of learning.

3.3 Back-Propagation

Back-propagation is the most commonly used method for training multilayer feed forward networks. The term back-propagation refers to two different things. First, it describes a method to calculate the derivatives of the network training error with respect to the weights by a clever application of the derivative chain-rule. Second, it describes a training algorithm, basically equivalent to gradient descent optimization, for using those derivatives to adjust the weights to minimize the error.

As a training algorithm, the purpose of back-propagation is to adjust the network weights so the network produces the desired output in response to every input pattern in a predetermined set of training patterns.

To describe the basic concept of back-propagation learning algorithm, each of its elements and how they are combined to form the back propagation topology is the first to be looked at. Figure (4) illustrates a simple three-layer feed forward neural network. In the figure it can be seen:

- Input layer A: The input vector I is feeding into layer A. It has L nodes, ai (I=1 to L), one node for each input variable.
- Hidden layer B: It has m nodes, b_j (j=1 to m).
- Output layer C: It has n nodes, C_k (k=1 to n), one node for each output variable: dk is the desired output, and c_k is the calculated output.
- Interconnecting weight between the ith node of layer A and the jth node of layer B is denoted as v_{ij}.
- Interconnecting weight between the jth node of layer B and the kth node of layer C is denoted as w_{ij}.
- Internal threshold value for layer A is TA_i, for layer B, TB_j, and for layer C, TC_k.

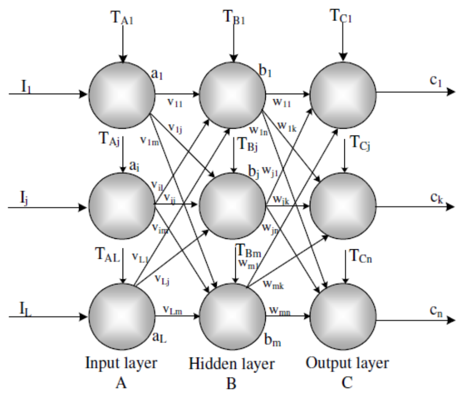


Fig.(4) Three-layer feed forward neural network.

4. Results and Discussion

Using programming techniques we have solved 165 Linear programming models (LP) analytically. Out of these 165 data sets we have taken 150 set for training the neural network and 15 set for simulating the network with new inputs to predict the output. The data set consist of eleven inputs variables viz. ,a_{11}, a_{12}, b_1, a_{22}, a_{22}, b_2, c_1, c_2 and three output variables viz., x_1, x_2,and Z. Table(1) shows few training data which was used to train the network. The network structure used to train the network consist of one input layer consisting of 11 neurons, and one hidden layer consisting of 165 neurons and one output layer having three neurons The training of the network was carried out with the neural network toolbox using Matlab. Many versions of Back-propagation training algorithms are available in which resilient. Back-propagation training is used which provides faster results.

Table(1): Data set used for training

Sample No.	Inputs											Outputs		
	a_{11}	a_{12}	b_1	a_{21}	a_{22}	b_2	a_{31}	a_{32}	b_3	c_1	c_2	x_1	x_2	z
1	2	-1	4	2	3	12	0	1	3	3	1	3	2	1
2	2	-1	8	2	3	10	1	0	3	1	3	0	3.33333	10
3	2	1	60	8	3	80				1	1	0	26.6667	26.667
4	0.5	1.5	6	0.8	0.33	8				2.5	0.8	10	0	25
5	1	1	16	8	-3	28				25	-8	6.909	9.0909	100
……	…	…	…	…	…	…	…	…	…	…	…	…	…	…
……	…	…	…	…	…	…	…	…	…	…	…	…	…	…
……	…	…	…	…	…	…	…	…	…	…	…	…	…	…
146	2	-1	40	1	1	53	0	1	8	1	1	45	8	53
147	2	-1	40	1	1	53	1	0	6	1	1	31	22	53
148	1	-1	40	1	3	53	0	1	2	3	7	47	2	155
149	2	8	33	9	7	49				10	19	2.7758	3.4310	92.9482
150	4	8	16	8	3	28				12	21	3.3846	0.3076	47.0769

We have used Back-propagation for training the network and simulating it with new inputs. There are generally four steps in the training process:

a. Assemble the training data set.
b Create the network object.
c. Train the network with sample data.
d Simulate the network with new inputs.

Properly trained networks tend to give reasonable answers when presented with inputs that they have never seen. The sample data for LP are assembled, trained and simulated with the network structured. The following is the algorithm for a three-layer neural network with one hidden layer. Initialize the weights in the network (often randomly)

```
i)
Do For each example data (e) in the training set
O = neural-net-output (network, e)
 forward pass, T = output for e
 Calculate (T - O) at the output units
 Compute delta_wi for all weights from hidden
layer to output layer ;
backward pass Compute delta wi for all weights
from input layer to hidden layer; backward pass
continued
Update the weights in the network
Until all dataset classified correctly or stopping
criterion satisfied
ii) Return the network.
```

Fifteen samples were taken for testing the ANN as shown in table(2). The analytical solution of these testing data set using classical methods was shown in table(3). In order to validate the performance of the ANN during the training process, the goal was set up to 0.01and it has taken 9278 epochs to train the network and the error performance was less than 0.0097, which shows the convergence. The trained network was then simulated using the 15 testing data set. The results were compared with the analytical results and were found to be different in fractional values which almost near to the analytical results. The results obtained by simulating the network are shown in Table(4).

Thus the network prediction of the solution to the LP was tremendous and with very little error percentage of range from (0.001497139-1.692335944)% for x_1 value, (0.000370370 - 1.219231219)% for x_2, and (0.004526984 - 2.705937143)% for z value, see table(5) . The comparison of the results of Analytical and ANN values of x_1, x_2 and Z values are shown in Figures 5,6 and 7 respectively. Hence the error percentage

sample No.	INPUTS										
	a_{11}	a_{12}	b_1	a_{21}	a_{22}	b_2	a_{31}	a_{32}	b_3	c_1	c_2
1	2	1	8	2	3	11				3	4
2	1	2	21	6	-2	12				7	2
3	2	1	14	1	1	3				9	1
4	1	-1	1	1	1	11				2	1
5	1	-1	4	2	1	11				1	2
6	1	1	22	1	3	38	2	-1	12	2	1
7	1	-1	10	1	3	34	1	-2	12	1	3
8	3	1	32	2	3	26	1	-2	6	2	-3
9	3	1	7	2	3	7	1	-1	1	2	3
10	3	1	63	2	3	49	1	1	23	0.5	-2
11	1	-1	63	2	3	74	1	-1	23	0.5	1
12	1	1	63	1	3	74	1	-1	23	0.4	1
13	1	-1	8	1	3	35				2	3
14	0.5	-0.6	8	2	3	35				1	1
15	0.5	-4	3	1	1	7				9	-3

is calculated using the formula:

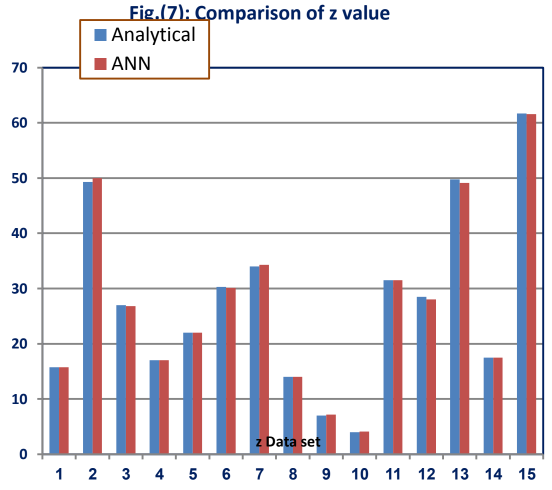
$$\textbf{Error}\ (\%) = \left(\frac{\textbf{Analytical value} - \textbf{ANN value}}{\textbf{Analytical value}}\right) \textbf{x 100\%}$$

Table(2): data set for testing

Table(3): Outputs of the analytical solution for data given in table(2)

sample No.	OUTPUTS		
	x_1	x_2	z
1	3.250000	1.500000	15.750000
2	4.714286	8.142857	49.285714
3	3.000000	0.000000	27.000000
4	6.000000	5.000000	17.000000
5	0.000000	11.000000	22.000000
6	10.571429	9.142857	30.285714
7	16.000000	6.000000	34.000000
8	10.000000	2.000000	14.000000
9	2.000000	1.000000	7.000000
10	10.000000	3.000000	4.000000
11	63.000000	0.000000	31.500000
12	57.500000	5.500000	28.500000
13	14.750000	6.750000	49.750000
14	17.500000	0.000000	17.500000
15	6.888889	0.111111	61.666667

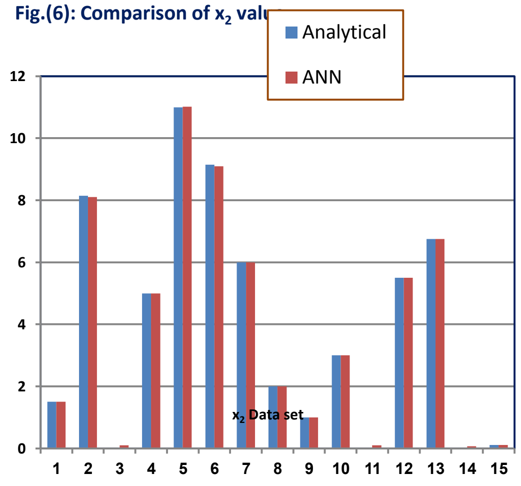
Fig.(7): Comparison of z value



Table(4): Outputs of the ANN solution for data given in table(2).

sample No.	OUTPUTS		
	x_1	x_2	z
1	3.246751	1.499356	15.749287
2	4.696475	8.100356	50.006567
3	3.017898	0.198645	26.797463
4	5.994678	4.993647	17.004534
5	0.234560	11.016374	22.018946
6	10.636819	9.100354	30.103847
7	16.001893	5.999647	34.276857
8	10.005821	2.004563	14.004563
9	2.0070950	1.000296	7.189415
10	10.019425	3.000465	4.098341
11	62.984251	0.095860	31.478655
12	57.472819	5.499354	28.000767
13	14.696217	6.749975	49.100008
14	17.500345	0.071498	17.463784
15	6.774246	0.112465	61.573829

Fig.(6): Comparison of x_2 value



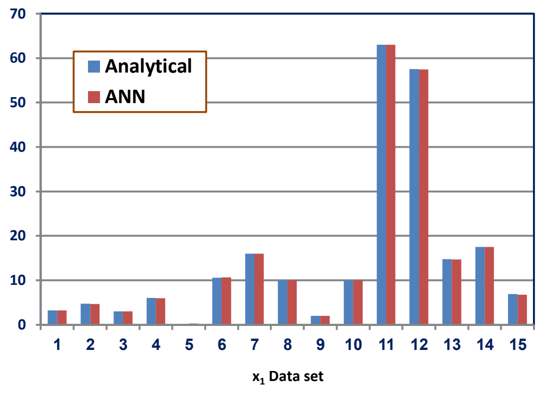


Fig.(5): Comparison of x_1 value

6. Conclusions

The analysis of the results obtained through ANN shows that it produces almost matching results as of the analytical method in solving out the Linear Programming Problem without the use of classical methods. Thus the implementation of the ANN in solving LP with neural network will produce efficient results.

References

[1]**FENG-TSE LIN**, "*A Genetic Algorithm for Linear Programming With Fuzzy Constraints", Journal of information science and engineering 24, 801-817* **(2008)** pp. 801-817.

[2] **S. Effati, M. Ranjbar**, "*Neural network models for solving the maximum flow problem", Applications and Applied Mathematics: An International Journal(AAM), Vol. 3, No. 1* **(June 2008)** pp. 149 – 164.

[3] **SezgiÖzen, G. MiracBayhan**, " *Optimization of Depth of Cut in Multi-pass Machining Using Hopfield Type Neural Networks", Proceedings of the 2011 International Conference on Industrial Engineering and Operations Management Kuala Lumpur, Malaysia*, **January 2011**, pp. 22 – 24.

[4] **M. Joorabian, R. Hooshmand**, "*Application of Artificial neural Network in Voltage and reactive power Control", Ph.D. thesis, Department of Electrical Engineering Shahid Chamran University, Iran,* **2009**.

[5] **J. AbdulSamath, P.Senthil Kumar, Ayisha Begum**, "*Solution of Linear Electrical Circuit Problem using Neural Networks", International Journal of Computer Applications (0975 –8887) Volume 2 –No.1,* **May 2010**.

[6] **Xiaolin Hu, Jun Wang**, "*Solving Generally Constrained Generalized Linear Variational Inequalities Using the General Projection Neural Networks", IEEE Transactions on Neural Network,Vol.18,No. 6,* **November 2007**, pp. 1697-1708.

[7] **Jerome K. Delson, S. M. Shahidehporv**, " *Linear Programming Applications to Power System Economics, Planning and Operations", Transactions on Power System, Vol. 7. No. 3*, **August 1992**. pp. 1155-1163.

[8] **Long Cheng, Zeng-Guang Hou, Min Tan**," *A Delayed Projection Neural Network for Solving Linear Variational Inequalities", IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 20, NO. 6*, **JUNE 2009**.

[9] **Stanislaw H. zak, Viriya Upatising, and Stefen Hui,** *"Solving Linear Programming Problems with Neural Networks: A Comparative Study", IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 6, NO. I ,* **JANUARY 1995**.

استخدام الشبكات العصبية الاصطناعية في حل نماذج البرمجة الخطية

د. عبدالكريم فليح حسن فتح الله فاضل خلف
قسم الهندسة الميكانيكية
كلية الهندسة – جامعة البصرة

الخلاصة

هدف البحث هو استخدام طريقة بديلة للطرائق التقليدية والتي عادة ماتستخدم في حل النماذج الرياضية الخطية والمتمثلة بطريقة التبسيط وتقنيات الحد والفرع. الطريقة المقترحة تمثل تطبيق نموذج شبكة عصبية اصطناعية باستخدام الحقيبة البرامجية MatLab . وجد ان استخدام الشبكات العصبية الاصطناعية تعطي نتائج مقبولة عند استخدام نماذج برمجة خطية مختلفة ،

يضاف الى ذلك تعتبر الشبكات العصبية الاصطناعية طريقة ذكية واكثر فعالية ،مقارنة بالطرائق الاخرى في نمذجة دوال الهدف المعقدة لاعتمادها على عملية التدريب حيث تعتبرهذه الطريقة اكثر طرائق الحل ملائمة عندما تكون كمية البيانات كبيرة (عدد المتغيرات والقيود) . هذا العمل يعرض ويناقش نموذج شبكة عصبية اصطناعية لحل نماذج البرمجة الخطية. نتائج النموذج اعطت تقديرا" جيدا" بنسب خطأ قليلة.

كلمات مرشدة: البرمجة الخطية ، الشبكات العصبية الاصطناعية.