Solution to the Tic-Tac-Toe Problem using Hamming Distance approach in a Neural Network

Nazneen Fatema Rajani
Dept. of Information Systems
BITS Pilani, K. K. Birla – Goa Campus
Goa, India
E Mail: emailnazneen@gmail.com

Dr. Gaurav Dar
Dept. of Physics
BITS Pilani, K. K. Birla – Goa Campus
Goa, India

Rajoshi Biswas
Dept. of Electronics and Instrumentation
BITS Pilani, K. K. Birla – Goa Campus
Goa, India

Dr. Ramesha C. K.
Dept. of Electrical, Electronics and Instrumentation
BITS Pilani, K. K. Birla – Goa Campus
Goa, India

***Abstract* — This paper focuses on using a Hamming Distance Classifier in Neural Networks to find the most optimal move to be made in the Tic-Tac-Toe problem such that the game always ends in a win or a draw.**

Keywords-Hamming Distance Classifier, Back Propagation Network.

I. INTRODUCTION

The Tic-Tac-Toe problem has been addressed in a number of ways to improve efficiency, accuracy and reduce complexity in the quickest method possible. However, using neural networks there are certain advantages to conventional approaches. Firstly, the machine tries to approach the problem the way human brain works but with no possibility of errors and quickly finding the most optimal ways to win or in some cases prevent loss. The usual approach is to use back-propagation algorithm [2] which minimizes the error calculated for each iteration to obtain the best move. In order to reduce the complexity of the design, the number of iterations, neurons, layers and certain other parameters like compactness of the code, the method of Hamming Distance Classifier is used.

II. ALGORITHM

The Hamming Distance (HD) between two strings of equal length is the number of positions at which the corresponding symbols are different. The concept used in this algorithm is Hamming Distance classifier, which selects the stored classes that are at minimum HD value to the noisy or incomplete argument vector presented at the input [1]. The Hamming distance algorithm uses two steps in order to give a best next move. This strategy either helps the network to win or else prevents the opponent from winning.

A. Step1:Basic Step

This step involves an eight-class Hamming network which has nine inputs and corresponding eight outputs. The eight classes are all the possible final winning configurations for the neural network or the opponent as shown in the Fig. 1:

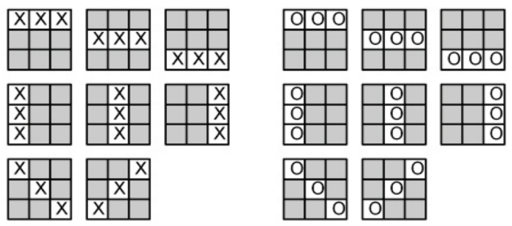

Figure 1. All winning configurations of the board.The opponent's move is X and network's move is O

The network will have nine inputs corresponding to each cell of the grid, and the input values will be limited to the following values, Fig.2:

0: Unoccupied cell
1: Protagonist player
-1: Opponent player

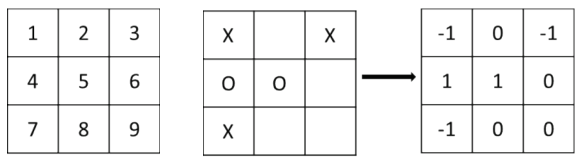

Figure 2. This shows the grid nomenclature.

The weight matrix consists of nine-tuple prototype vector of each of the eight winning classes.

$$W = \begin{bmatrix} 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & -1 & 1 & -1 & -1 \\ -1 & 1 & -1 & -1 & 1 & -1 & -1 & 1 & -1 \\ -1 & -1 & 1 & -1 & -1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & 1 \\ -1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & -1 \end{bmatrix} \quad (1)$$

The above weight matrix is used to design the network as follows, Fig. 3:

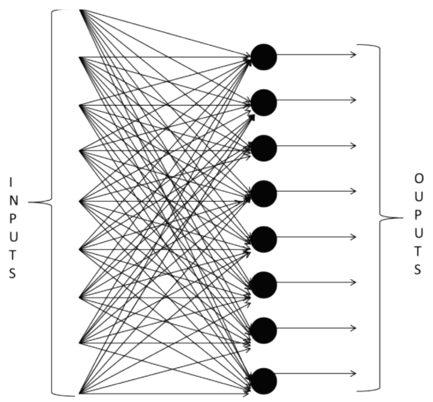

Figure 3. Network Diagram

Using this design of the network, an input is given to the Hamming network with an activation function which performs a linear scaling. The input vector Y contains the current configuration of the board. A fixed bias value of 9/2 is added to input of each neuron to yield

$$Net = \frac{1}{2}WY + C\ , \tag{2}$$

Where C is the column vector of fixed biases.

Since the output of the above formula yields a Hamming distance between 0 and 9, it is scaled down to lie between 0 and 1 using

$$F = \ Net/9 \tag{3}$$

Two other arrays are extracted from Y for winning configurations of X and O respectively as follows:

X array: each of the -1 in input array is replaced with 1 and rest all are replaced with a -1, Fig. 4(b).

O array: each of the 0 in input array is replaced with -1 and rest all remain same Fig. 4(b).

When these arrays are given as an input to the Hamming distance network sequentially, we get two output arrays of length 8 each, we call them output of X and output of O. These represent the hamming distances of input from winning configurations for the opponent and the neural network respectively. Using these 16 values the network makes a plausible move as follows:

X		X
O	O	
X		

➡

-1	0	-1
1	1	0
-1	0	0

(a)

1	-1	1
-1	-1	-1
1	-1	-1

-1	-1	-1
1	1	-1
-1	-1	-1

(b)

Figure 4. (a) Input configuration.(b) Extracted X and O arrays respectively.

1. The maximum of the 16 output values say ‘i’, is selected which indicates the class at the smallest Hamming distance.

2. If the value belongs to the first set that is output of X, the network compares the values represented by the ith row of the weight matrix with the current board configuration and checks if there is a chance of opponent winning in the opponent’s very next move, if so the network prevents it by taking a move along that configuration.

3. Else if the value belongs to the second set that is output of O, again the network compares the values represented by the ith row of the weight matrix with the current board configuration and checks if the configuration along that direction contains either 0 or 1. If so, it makes a move to the first unoccupied position along that configuration. Thus, it moves a step closer to winning.

4. If none of the above two conditions satisfy, the network replaces i by a negative number like -1 so that it is not selected again. Now, the network proceeds to find the next maximum and repeats the above steps till a satisfactory move is decided.

In the case where the next move is a winning move for both the network and the opponent, the Hamming distances calculated are such that the network prefers to win rather than avoiding the opponent from winning.

B. Step 2. Iterative step

In this step, the neural network anticipates the next move of the opponent which helps it strengthen its decision regarding the current move to be made. The algorithm proceeds as follows:

Initially the network maintains an array of hamming distances which helps it decide on the best possible next move. This array is updated in the following manner:

1. Given a current input configuration, the network calls the basic step to predict the next move say ‘k’. Let HD1 be the hamming distance array for the basic step. HD1 (k) is the maximum hamming distance of the array currently.

2. Once the next move is decided, the network runs the basic step again for each possible opponent’s next move based on the network’s current move. This produces the next to next move of the network for each possible next move say ‘m’, of the opponent. This produces m maximum hamming distances for each possible next move which are then added to HD1 (k).

3. The network replaces the next move in step 1, by second maximum hamming distance move and repeats the above steps. Then the network replaces the next move with third maximum hamming distance move and repeats the above steps. This continues till all possible next moves are exhausted.

4. HD1 now contains the gross hamming distance due to all iterations. The maximum of HD1 gives most profitable next move.

This next move is final decision of the neural network based on the opponent's previous move. The above algorithm is executed every time it is the network's turn to make a move.

III. RESULTS

The Hamming Distance algorithm was coded using MATLAB. The simulation results for certain cases of inputs are shown below:

1. The following configuration, Fig. 5(a) is given as input to the network. Here, in the next step it is possible for neural network to win horizontally along the second row, as well as prevent the opponent from winning horizontally along the first row, but the network prefers to win by choosing the next move as cell number six as shown in Fig. 5(b).

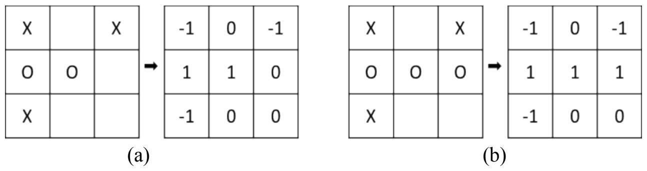

Figure 5. (a) The Input configuration. (b) The output configuration.

2. If we give an input such that the opponent is about to win say diagonally as in the given configuration, Fig. 6(a), the neural network prevents it from winning by correctly making the next move to cell number seven, Fig.6(b).

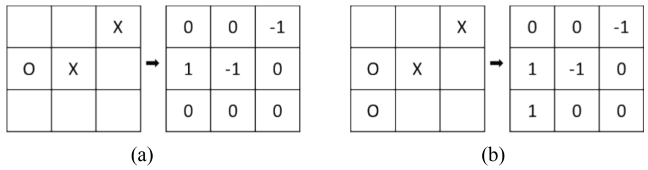

Figure 6. (a) The Input configuration. (b) The output configuration.

IV. DISCUSSION

The Tic-Tac-Toe game has been implemented using neural networks previously but it was done using error back-propagation algorithm. In this algorithm, the network undergoes learning using inputs and their corresponding desired outputs. Error is calculated at each step by comparing the expected and actual output and weights are updated to reduce this error. This algorithm is quite inefficient because of following reasons. Firstly, the neural network is trained using every possible board configuration (input) and the corresponding best move (desired outputs). Secondly, the network is to be initiated using random weights which are updated at each learning step, it is highly probable that the weight combination may reach a local minima at the error surface and get stuck. As a result, the simulation needs to be restarted with new random weights. Thirdly, since the learning rate determines how much influence error values have on weight and bias changes, it has to be wisely chosen. Finally, a high efficiency game which can predict best moves and win the game requires hidden layers with a large number of neurons and hundreds of iterations which is computationally complex [4]. It uses 48 neurons in the first layer, nine in the second and runs for 200 iterations with variable weights, to find the most optimal move.

However, the Hamming Distance algorithm proposed by us uses only eight neurons with fixed weights that do not change at each step. This algorithm runs for 100 iterations and also has comparatively much lesser runtime. Since the network is scalable, the calculated time complexity of the algorithm is O (n^3) where n is the number of cells (here nine).

V. ADVANTAGES OF THE ALGORITHM

The basic step itself is sufficiently accurate. The iterations are used to make the code more efficient. The efficiency of the code is such that the game always ends in a win or a draw. Since the basic step is the network of fixed weights with eight neurons, the hardware implementation has reduced tremendously. For a reasonably efficient game (using only basic step) one can implement the network using analog electronics, where each neuron will be replaced by an operational amplifier and each weight by resistors. The circuit is very simple, compact and inexpensive. Since weights in error back-propagation algorithm keep updating at each step, it is not possible to implement it using analog electronics.

The iterative step needs to be implemented for higher efficiency using digital electronics. The loops and conditional statements (which cannot be designed using analog electronics) can be simulated using digital platforms like Field Programmable Gate Arrays (F.P.G.A) and can be built on Application Specific Integrated Circuits (ASICs). The advantage of using this network is that the basic unit is very compact and the only extra (optional) requirement is the implementation of the iterations.

Since the software program of the algorithm is compact and simple it can be implemented as applications for mobile and other portable devices.

VI. CONCLUSION

In this paper, we have looked into the usage of Hamming Distance Classifier to solve common problems like Tic-Tac-Toe. This is important because there are new algorithms developing every day, thus the need for optimal and less complex algorithms has become vital. Our contributions to the Tic-Tac-Toe problem can be summarized as follows:

1. Considering different approaches to the problem
2. Developing an optimal algorithm to solve the problem.
3. Improving the efficiency of the algorithm while maintaining polynomial time complexity.

This research may be helpful in improving the complexity and providing optimal solutions to similar problems.

FUTURE WORK: The algorithm proposed by us uses iterations some of which are redundant. We can exploit the symmetry of the input configuration to eliminate those cases and thus reduce the number of iterations drastically.

Also, this paper deals only with the software aspect of the algorithm. We plan to simulate the algorithm using VHDL on F.P.G.As.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable comments. We also thank all those who have helped in this work.

REFERENCES

[1] Jacek M. Zurada, " Introduction to Artificial Neural Systems," West Publishing Company,2006.

[2] Simon Haykin, "Neural Networks A Comprehensive Foundation," Pearson Education, 2nd ed, pp 159-178.

[3] Patrick H. Winston," Artificial Intelligence, Addison-Wesley, 3rd ed,1993.

[4] Neural Network with learning by backward error propagation. Available: http://www.colinfahey.com/

[5] K. Pavel, K.Jan , D. Jan, K. Oleg, Č. Miroslav, and Š. Miroslav, "Meta-learning approach to neural network optimization," Neural Networks, vol. 23, issue 4, pp.568-582, May 2010.

[6] Terence D. Sanger, "Optimal unsupervised learning in a single-layer linear feedforward neural network ," Neural Networks, vol. 2, issue 6, pp. 459-473,1989.