Designing a 5x5x5 Tic-Tac-Toe Game using a Neural Network with Backpropagation with a Twist

Shahzeb Siddiqui
sms5713@psu.edu

Francis Mutuc
fcm5007@psu.edu

Nicholas Schmidt
nas246@psu.edu

Department of Computer Science & Engineering
Penn State University Park
201 Old Main, University Park, State College, PA
(814) 865-9118

Abstract*—Tic-Tac-Toe is a game which has been explained in great detail. It is a two player game with one player who plays as crosses 'X' and the other as noughts 'O'. In the tradition 3x3 game, the primary goal is to win via three in a row. If this goal is unreachable, then it becomes necessary to force a draw. However, our current Tic-Tac-Toe board has two additional rows, two additional columns, and has been extended to the 3rd dimension. As such, the goal of this game is to get 5 in a row either vertically, horizontally, or diagonally. The 5x5x5 Tic-Tac-Toe game consists of five planes each containing 25 tiles to have a total of 125 tiles. With the addition of another dimension comes increased complexity, since each player can now make 5 in a row across planes. In this paper, we will discuss how to implement an artificially intelligent agent to play the proposed 5x5x5 Tic-Tac-Toe game. We propose the idea of creating a neural network that uses backpropagation coupled with elements of a genetic algorithm to improve the likelihood that the most optimal solution is obtained and outline our methodology at the implementation level.

***Keywords*—**Neural Networks, Tic-Tac-Toe, AI, Back-Propagation

I. INTRODUCTION

Tic-Tac-Toe is a well-known game that can be found anywhere such as computers, mobile apps, and video games. Tic-Tac-Toe is a topic of research in the field of artificial intelligence (AI) because it can be shown to exhibit learning by playing against other players. Our goal is to design an AI agent that can successfully compete against an experienced player by learning the opponent's strategies. Ideally, our agent must maximize its win-loss ratio as well as its win-draw ratio. Our agent will gradually learn how to play the game using patterns of the board that we deem are good moves. Because of its inherent accuracy at pattern matching, neural networks have been used in many applications including Tic-Tac-Toe and other games as well as fingerprint matching [8]. However, the learning aspect of the neural network technique comes in the training algorithm, where the neural network is conditioned to learn patterns based on a reward/punishment basis. There are many techniques present concerning the art of training a neural network, but the most common implementation is to use what is called backpropagation. Backpropagation is a technique where the neural network will randomly play a move and learn from its mistakes by modifying its network connections using values called weights and propagating those changes backward from the output layer through the hidden layer eventually reaching the input layer. In analyzing backpropagation, we have decided to modify backpropagation in our neural network to incorporate some aspects of a genetic algorithm. We will explain how we will try to implement this idea later in the paper.

II. WHAT IS A NEURAL NETWORK?

A neural network is a network consisting of nodes called neurons which are linked together by a synapse. It was conceptually designed to mimic the network connection of a human brain. A neuron is the basis for information processing. A link connects two neurons together. Each link is attached a weight which determines the strength of the link. The weight of each link is modified every time a player makes a move. The neural networks consist of three distinct layers: an input layer, a hidden layer, and an output layer. The input layer consists of input neurons. Hidden layer is a variable number of layers each consisting of neurons which modifies the input to form a distinct output. In our research, we have concluded that a good heuristic to choosing the number of hidden layers is to not exceed 3 hidden layers because of complexity in its implementation. Output layer determines state of neurons at output.

III. ADVANTAGES OF NEURAL NETWORKS OVER OTHER TECHNIQUES

A neural network is an effective way to implement an agent to actively learn how to play Tic-Tac-Toe because the agent learns from its moves every game. A rule-based expert system is another method for implementing AI. Unfortunately, the rule-based expert system only plays by the rules assigned in the knowledge base. There is no learning involved because the rules are pre-programmed into the knowledge base. One problem that a rule-based expert system faces is when a rule is not covered by in its rules then the AI will do a random move, which ideally should be avoided.

Another way to implement artificial intelligence is by using a genetic algorithm (GA). GAs are used for optimization. However, the problem with genetic algorithms is that they would have a hard time identifying the difference between a local maximum and a global maximum. It is very common to see a genetic algorithm/neural network hybrid; research has shown that both techniques work well with each other because the neural network narrows the sample space of the genetic algorithm which allows for a better global maximum in its region.

IV. BACKGROUND

Colin Fahey demonstrates how to use a neural network with backward error propagation in order to produce an AI agent that learns how to play tic-tac-toe. Fahey outlines the structure for the neuron body and neuron link with all of its interconnections. During the training process, there is a list of errors Fahey mentions that can occur:

1. Weight combination has reached a local minimum
Solution: Restart simulation with random weights

2. Network has too few neurons or layers to encode all the patterns in the training set.
Solution: Add neurons in hidden layer, or extra hidden layer
3. Items in the training set contradicts with the entire set which skews the data
Solution: Find test items that yield the greatest error deviation and modify the test item by using weighted average to eliminate deviation

4. Learning rate is too high (usually greater than 1.0) which will cause AI to over-learn.
Solution: Reduce learning rate

5. Learning rate is too low (usually 0.01) which will cause network to converge to an ideal value. This causes AI to learn very slow and training will have no effect
Solution: Increase learning rate

The training simulation that Fahey conducted was the overall root-mean square (RMS) error of all training items. This result is effective in characterizing the network overall error for all training simulation. The root mean square calculation can be performed in two steps: [3]

$$\text{squaredError} = \sum_{i=1}^{TotalTrainingItems} (Output.Error)^2 \quad (1)$$

$$rmsError = \frac{squaredError}{TotalTrainingItems} \quad (2)$$

Kumar investigated Tic-Tac-Toe game using Neural Networks, Rule-based and Evolutionary Learning. Kumar used similar concepts from other scholars when designing AI system for Tic-Tac-Toe game. He used the sigmoid function, with the same scoring mechanism (-1, 0, 1) for “0” empty space and “X”.

His Neural network was designed with 9 input and output neuron with a single hidden layer. During the learning he used genetic algorithm by using 50 parent nodes with variable number of neurons in hidden layer between [1, 10]. Each weight link was initialized randomly with bias set to zero. The weights were between (-0.5, 0.5) with Gaussian mean = 0, standard deviation = 0.05. During learning, the AI will receive a new board configuration as its input for the neural network.

Kumar described three payoff functions {+1,-1, 0}, {+1, -10, 0} and {+10,-1, 0} where the entries are winning, losing, and draw. Kumar tested four sets of game with 8 trails totaling 32 simulation rounds where the maximum score can be 32 for the first and second payoff function and 320 for the third payoff.

The learning rate with payoff {+1,-1, 0} resulted in consistent with all trials with 95% confidence level. The payoff function {+1,-10, 0} has a different behavior than the 1st payoff function because penalty for losing is higher which increased the initial learning rate quite rapidly. Strategies that lose were eliminated from the population set during the learning process. The benefit of this payoff function across 32 trails is that the AI never caused the opponent to win in any of the games. Most games resulted in a draw or win for the AI. Finally {+10,-1, 0} payoff function had similar results as the first payoff function {+1,-1, 0}. The only difference is that winning received 10 more points than a draw. After 800 generations of simulation using GA, the only surviving strategies were winning or draw strategy. All losing strategies were eliminated in the learning process. [7]

V. SCORING TECHNIQUE

Most Tic-Tac-Toe games using a neural network have a scoring mechanism for each move. The Tic-Tac-Toe game will incorporate a score value for X, O and blanks. An ‘X’ will have a score value of 1, an ‘O’ will have a score value of -1 and the blank will be 0. This scoring mechanism will be the inputs into the neurons.

VI. ACTIVATION FUNCTION

The activation function is the mechanism by which input is propagated in the forward direction in the neural network. For our activation function, we will use is the sigmoid function

$$Y_{sig}(X) = \frac{1}{1+e^{-X}} \qquad (3)$$

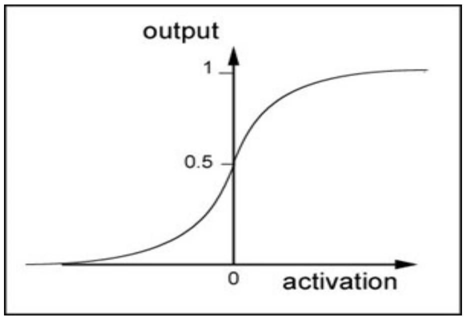

Figure 1: Sigmoid Function

We chose to use the sigmoid function because we want the output layer to have values between (0, 1). The sigmoid function according to figure 1 is between (0, 1) for x all values. Our agent will move in the location with the neuron with the highest score value. The output vales are defined as:

0: Do not move here
1: Move here

VII. HEURISTIC INPUT FILTER

We have decided to implement a heuristic to search for a plane with the best situation of good moves in order to reduce our input neurons and computation within the neural network. We have a strategy implemented that determines what a good move is. In this strategy, we give highest priority to taking a move that directly results in a win i.e. creating a 5 in a row. Then, the next highest priority goes to preventing the opponent from winning i.e. blocking an opponent from creating a 5 in a row. Next, we give priority to a scenario where we have 3 in a row, causing us to create a 4 in a row. Afterwards, we look to see if our opponent has 3 in a row and we subsequently move to block him/her from creating a 4 in a row. The next two scenarios will be if we have 2 in a row to create 3 in a row, similarly we block an opponent’s attempt at getting 3 in a row.

Every time an X is placed in a tile, we increase the score by 1 in all possible winning combinations in line with the player’s move. Likewise, when opponents place an O in a tile, we decrease the score by 1 in all possible winning combination in line with the opponent’s move. Initially, the game board is set to 0 because board is empty; the values will change as the game progresses. The search heuristic will provide a 25 tile plane as an input into the neural network.

VIII. DESIGN

We will have 25 input neurons and 25 output neurons. The 25 neurons would map to the 25 tile plane from the search heuristic. Initially, the weights for each link into the hidden layer will be assigned randomly between [0, 0.5] for training the AI. The input function for each neuron in hidden layer known as input accumulator is

$$I(X_o) = \sum_{i=1}^{n} x_i(p) w_i(p) \qquad (3)$$

X_i is the input neuron connected to neuron X_o, and w_i is the weight between X_i and X_o. N is the number of links connected to each neuron X_o.

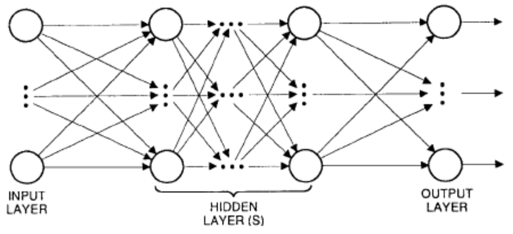

Figure 2: Multi-layer Network with Input layer, Hidden layer and Output Layer [7]

Once the search heuristic provides input to the neural network, the input neurons will propagate towards the output layer. By the output layer the AI will make a move in the location of its choosing. This process is repeated until the game is over. After completing a game, the weights would change depending on the board configuration.

IX. HIDDEN LAYER

In our 3-dimensional game, we will define x as columns, y as the rows, and z as each 25 tile plane.

Our neural network will consist of 1 hidden layer of 25 neurons. The 25 neurons will modify the 25 input neurons. Each of the 25 neurons in hidden layer will be a weighted sum of the 25 input neurons. The goal in this hidden layer is assess the strength of each search plane to determine where to place a move. This is done by analyzing the weights, i.e. a higher weight defines a move that we will make.

Our Neural Network design is shown in the figure below

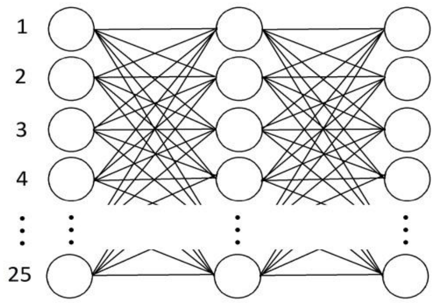

Figure 3: Neural Network

X. NEURON BODY

The body of the neuron will compute the output and the error-accumulation and the bias given the input signal in (3). The output of the neuron is defined by this equation:

$$Output = Y_{sig}(bias + InputAccumulator) \tag{4}$$

Y_{sig} is the activation function defined in (10) The Input-Accumulator is defined in (1) which is the weighted sum of each neuron and the link. The bias equation is:

$$Bias \mathrel{+}= (-1) * Rate * ErrorAccumulator \tag{5}$$

The bias is an adjustable value that changes the input value before it gets passed into the activation function. The rate is the learning rate α. The error-accumulator is a numeric value which represents the error deviation in the neuron body after back-propagation. The equation is

$$ErrorAccumulator \mathrel{+}= Output * (1 - Output) OutputError \tag{6}$$

The OutputError is the error resulting from the next layer in the back propagation.

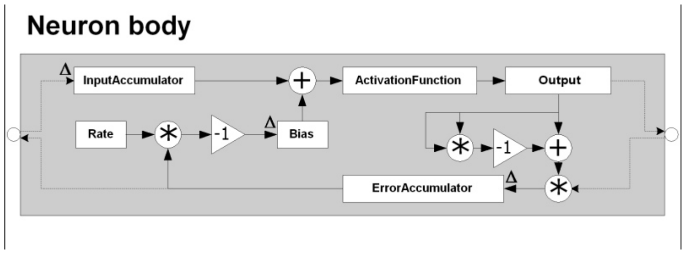

Figure 3: Neuron Body [3]

XI. NEURON LINK

The neuron link represents a connection between two neuron bodies. The neuron link has an associated input, output, and weight. The input to the link is the output from the starting neuron. The output of the link is the input to the ending neuron. The weight determines how the signal gets transmitted between the input and output state of the link.

The output of the link is:

$$Output = Weight * Input \quad (7)$$

The error in the link is determined by the weight factor or the Weighted-Error. The Weighted-Error function is

$$WeightedError = Weight * Error \quad (8)$$

The weight is adjusted according to the Learning Rate, Error and the Input value for the link to minimize WeightedError. The equation for weight is:

$$Weight \mathrel{+}= (-1) * Rate * Input * Error \quad (9)$$

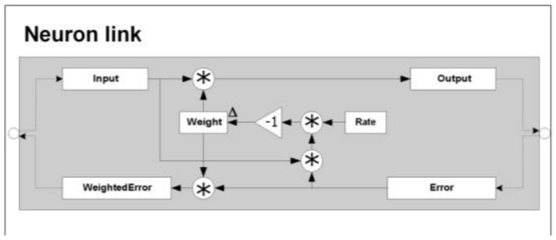

Figure 4: Neuron Link [3]

XII. LEARNING RATE

Every time the AI wins we will increase the learning rate α by 10% and we will decrease the learning rate every time the AI loses by 5%. However, we will have a min and max learning rate value defined set as: $\alpha_{min} = 0.1,\ \alpha_{max} = 0.8$

The min and max learning rate will be important for the AI to be in a safe region where it will always learn whether it is losing or winning at a significant rate. If learning rate is too low, then the agent would not have any significant change because $\Delta w_i(p)$ wouldn't change. However, if the learning rate is increased the too much ($\alpha > .8$), then $\Delta w_i(p)$ would be a large number causing in-proper weight calculation. Overlearning can cause overfitting and would result in the agent only learning specific patterns and not more general patterns.

XIII.BACKPROPAGATION AND THEN SOME

As stated earlier, we will amalgamate aspects of genetic algorithms into our backpropagation. The shortcoming to backpropagation is choosing which weights are rewarded and which weights are punished. If poor choices in doling out such outcomes are made, then the network takes much longer to learn the given patterns. In an effort to overcome this problem, we researched and have found several solutions wherein a genetic algorithm was used as a hybrid solution. However, we will not be implementing a full genetic algorithm. We will use it as a means to an end. We will attempt to use the genetic algorithm in order to better generate which connections are rewarded and which are punished. Our idea would be to have our weights be our competing elements. Then, we would have weights compete for at most 50 times. This number was chosen because we want to do this routine many times and a larger number for competitions would require more computation steps. After the competition, the top 5 are selected and have their value increased by 0.01 to simulate a mutation. The final part of the genetic implementation is we have this process repeat for 100 times to ensure that the weights are all unique. Again, we chose this number to reduce the number of computation steps involved. Afterwards, we backpropagate these new weights using the root mean squared.

XIV. TRAINING ALGORITHM

In research, we have found that games like our Tic-Tac-Toe game can be encoded in a rule-based way and found that a well implemented rule-based expert system agent is a fairly good player, having the probability to cause winning scenarios to be highly likely. In that regard, we have decided to try to have our neural network train against this kind of agent in the hopes of learning overarching strategies for winning. We know of several groups that are implementing a rule-based expert system as their AI agent and plan to speak with them so that we may use their agent to train our agent. If we cannot find a group willing to help us, then we will resort to having the neural network compete against itself for an arbitrarily high number of iterations (possibly 500).

XV. CONCLUSION

In the hopes of design an artificially intelligent agent that would play Tic-Tac-Toe at a competent level comparable to a human's level of competence, we have sought and sift through much research. We have found many interesting solutions, some of which are hybrid solutions, and have decided to propose a neural network agent that uses elements of a genetic algorithm design in its backpropagation. The method behind our madness in choosing to design a neural network as our agent is the learning aspect of neural networks was greatly seductive. We want our agent to exhibit something analogous to learning, a trait that we only found in the neural network technique.

Yet, there were other techniques that also seduced us. In looking at other solutions, we saw the genetic algorithm technique as an interesting approach. As such, we have aspects of it in our backpropagation. Using the design mention earlier in the paper, we hope to create an agent that would strive to player as well as a human would.

REFERENCES

[1] Schleis, G.; Rizki, M.; , "Learning from a random player using the reference neuron model," Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on , vol.1, no., pp.747-752, 12-17 May 2002 doi: 10.1109/CEC.2002.1007019
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1007019&isnumber=21693

[2] Fogel, D.B.; , "Using evolutionary programing to create neural networks that are capable of playing tic-tac toe," Neural Networks, 1993., IEEE International Conference on , vol., no., pp.875-880 vol.2, 1993
doi: 10.1109/ICNN.1993.298673
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=298673&isnumber=7404

[3] Fahey, Colin, *Neural network with learning by backward error propagation*
URL: http://www.colinfahey.com/neural_network_with_back_propagation_learning/neural_network_with_back_propagation_learning_en.html

[4] Siegel, Sebastian, *Training an Artificial neural network to play Tic-Tac-Toe*

[5] Pilgrim, Robert, *TIC-TAC-TOE: Introducing Expert Systems to Middle School*

[6] Negnevitsky, Michael, "Artificial Intelligence A Guide to Intelligent Systems"

[7] Chellapilla, K.; Fogel, D.B.; , "Evolution, neural networks, games, and intelligence," Proceedings of the IEEE , vol.87, no.9, pp.1471-1496, Sep 1999
doi: 10.1109/5.784222
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=784222&isnumber=17008

[8] Ammar, H.H.; Zhouhui Miao; , "Implementation of a training set parallel algorithm for an automated fingerprint image comparison system," Parallel Processing, 1996., Proceedings of the 1996 International Conference on , vol.2, no., pp.50-53 vol.2, 12-16 Aug 1996
doi: 10.1109/ICPP.1996.537381
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=537381&isnumber=11247