*Neurogammon*

Harrison Bonner   
Computer Science Dept.  
Indiana University Purdue University IndianapolisIndianapolis, United States of America  
hbonner@iu.edu  
  
Truman Brubaker  
Computer Science Dept.  
Indiana University Purdue University IndianapolisIndianapolis, United States of America  
trubruba@iu.edu  
  
Zachary Caswell  
Computer Science Dept.  
Indiana University Purdue University IndianapolisIndianapolis, United States of America  
zcaswell@iu.edu  
  
Ahmet Cengiz   
Computer Science Dept.  
Indiana University Purdue University IndianapolisIndianapolis, United States of America  
ahcengiz@iu.edu

Sabrina Chowdhury   
Computer Science Dept.  
Indiana University Purdue University IndianapolisIndianapolis, United States of America  
sabchow@iu.edu

Foster Clinton  
Computer Science Dept.  
Indiana University Purdue University IndianapolisIndianapolis, United States of America  
fclinton@iu.edu

Matthew Cobbs  
Computer Science Dept.  
Indiana University Purdue University IndianapolisIndianapolis, United States of America  
cobbsma@iu.edu

Nick Derf  
Computer Science Dept.  
Indiana University Purdue University IndianapolisIndianapolis, United States of America  
nderf@iu.edu

***Abstract*— Neurogammon 1.0 was a groundbreaking achievement when it won the First Computer Olympiad in London, England, Nineteen Eighty-nine.  The purpose of this paper is to provide a comprehensive review of the research conducted by our group on how a multi-layer neural network that uses back propagation, similar to Neurogammon, operates.  Concluding remarks and possible directions of growth are also discussed at the end.**

***Keywords—Neural Network, Game Theory, Artificial Intelligence, Backgammon, Computer games, Learning Systems***

# Introduction

Backgammon at the basic level is a game where two adversarial players take turns rolling dice and moving their pieces. The game is won once a player has successfully moved all of their pieces (singularly referred to as a checker) off the board. The game of Backgammon, however, can become extremely complex once all of the rules are factored in.  One of the main rules that affects the outcome of the game is through the process of having a checker land on a blot (what an opponent's checker is referred to).  When a hit occurs, the blot that is landed on is immediately sent back to the starting point of the opponent's game board.  This hinders the progress of the opponent’s other blots by not allowing them to move until the blot that was hit has moved from the starting point.  Expert players take advantage of the hit rule to set up complex blocking techniques that render their opponents unable to move.  The complexity of the game can be increased further through the use of a cube that allows you to gamble on whether or not you want to double the stakes of the current match.  These are the defining rules in backgammon that allow professional players to develop advanced strategies to use against their opponents [1][4]. Due to the highly non-deterministic nature of the game, it is a good candidate for a neural network as an artificial intelligence opponent.

# History and background

Gerald Tesauro is currently a Principal Research Staff Member in AI Science at IBM’s TJ Watson Research Center. He is also a Fellow of the AAAI (Association for the Advancement of Artificial Intelligence), a Fellow of the ACM (Association for Computing Machinery), a member of the Board of Directors of the NeurIPS (Neural Information Processing Systems Foundation), and an Associate Editor of the ICGA (International Computer Games Association) Journal. Tesauro is most famous for his Neurogammon program, an AI Neural Network that learned how to play Backgammon and won the First Computer Olympiad, an Olympic style event where computer programs compete against each other for the title of world's best, that occured in nineteen eighty-nine.

 Gerald Tesauro first published an article in 1986 called “Simple Neural Models of Classical Conditioning”[5], which then paved the way for his 1987 publication on the game of Backgammon. Two years later, in 1989, Tesauro’s research allowed his AI to win the Firstt Computer Olympiad in London after defeating all five opponents that the program was pitted against in Backgammon. At the time, the Neurogammon program was able to play at an intermediate level of skill when evaluated in terms of human capabilities.

The first version of the AI, which was called Neurogammon, consisted of a neural network that was trained by using backpropagation on a pre-recorded data set of expert games. The inputs used to train the program included the board information and the set of features. This was the version that competed in the Computer Olympiad in London. It not only won, but it claimed five straight victories.

Tesauro then continued his research and went on to create another, improved version of this AI.  This version he called TD-Gammon. Rather than the backpropagation techniques used by Neurogammon, TD-Gamm instead used reinforcement learning through multilayer neural networks. These neural networks were trained by Temporal Difference (TD) so that it was able to learn and implement more complex, nonlinear functions.

Tesauro’s main field of interest is neural networks and data modeling, even still to this day. One of his most recent publications in 2018 is, again, about Neural Networks and it is titled: “R^3: Reinforced Ranker-Reader for Open-Domain Question Answering.” [3]

# Neural Network Training

## Introduction To Neural Networks

Neural networks endeavor to recognize relationships between a set of data through a systematic process. It is designed in such a way that it can mimic biological neurons. In other words, it has simulated neurons that replicate human thinking processes. Neural networks are generally multilayer to classify series of data. It generally has three layers: the Input Layer, the Hidden Layer, and the Output Layer. Consider the single neuron pictured in figure 1. X is our input layer and B1 is the slope estimator of logistic regression. In the hidden layer, the sigmoid function is used for activation. Assuming B0 is the network’s bias, the outputted predicted probability is equal to the sigmoid function of B1\*x+B0

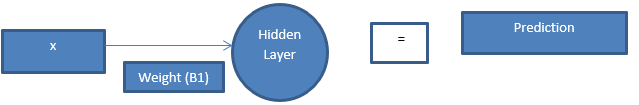


Figure 1- Single Neural Network Neuron

When this neuron is expanded for multilayer, the network looks something like Figure 2. There are two inputs and two neurons to support the inputs, and WX,Y is a weight associated with input X and neuron Y. The output of the hidden layer will be calculated as: Z1=W1,1\*In1+W2,1\*In2+bias\_n1. Neuron 1 activation =sigmoid (Z1). So, we can summarize this as:

A close up of a logo

Description automatically generated

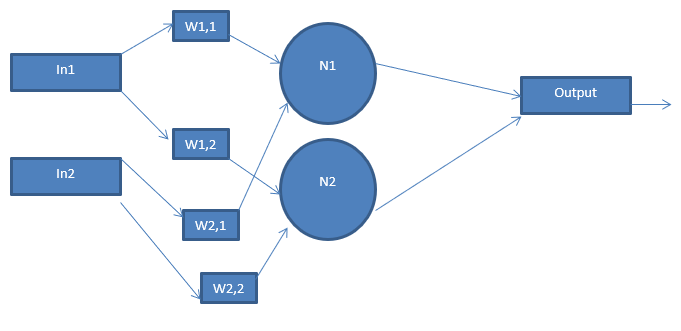


Figure 2- Multilayer Neural Network

## Training Data

For our multi-layer neural network to successfully function, we must first train it from some generated data. Our model utilizes thirty features: number of checkers in each position of the board (twenty-four features), dice roll (two features, one per die), number of checkers in jail (two features, one per player), number of checkers in the home (two features, one per player). Our data originated by capturing the thirty features [FC1] for some observed games. We accounted for the adversarial nature of the game by using negative numbers for the opposing player.

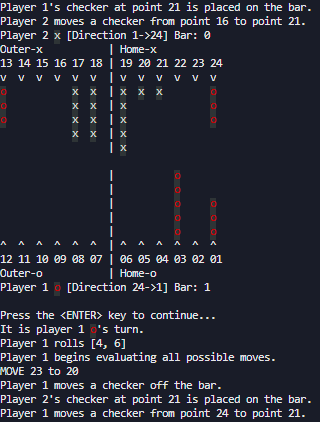
## Neural Network Layout

As described in the multilayer neural network introduction, our neural network will consist of thirty-one inputs (one for each feature) and five hidden layers. The output of the neural network will be classified using the sigmoid function, which will  output a binary zero for a current player loss, or one for a current player win.

D. *Results*

For this project, the game backgammon was written in python and a simple GUI developed in the console to observe that the game behaved as expected.   Control of each player was given to a separate agent.

Data was generated by having the two agents play each other continuously.  Each time a dice was rolled, the agent would evaluate each resultant board state from all possible moves.  Each move was given a different weight determined by the aggregate of how many times the resultant board state led to victory or defeat.  The move with the highest chance of victory was chosen as the next move.



At the conclusion of a game, training data was either created (if none existed for that board state) or updated with the outcome of that game.  During the training process millions of different board states were recorded.  The data for each individual board state was placed into its own bucket (file).  In which necessary records could be instantly retrieved and could be used for comparison as needed.

With each successive playthrough each agent gained more data into which moves were the most optimal.  The goal is to have a fully trained agent, with millions of games played, square off against an agent with no training data.  The expected outcome was that the trained agent would win most of these matches.

# *Results*

## Training Statistics

Once the infrastructure was in place and a significant amount of training data existed, the neurogammon self-learning was run 150 times. The winning player (both agents), number of turns, and trained states were tracked. Then analysis done with IBM SPSS.

Player 2 won more often at a surprising rate of 75.3%. In fact, using Bayesian One-Sample Binomial  Distribution, the win rate of the 2nd player with a 95% confidence interval is between 67.5% and 81.2%.  This high win rate suggests that being the second player may give an inherent advantage to the game, however it could be due to the current state of the training data causing some skewing to the 2nd player’s advantage.

The number of turns needed for either player to win averaged 64 turns with a standard deviation of 11.6 turns This suggests that with a 95% confidence interval that on average a game will last between 62 and 66 turns.

A close up of a map

Description automatically generated

Even with 48,000+ states known, the number of states learned from each game continued to grow, so an analysis was done to the number of states to attempt to estimate the maximum number of configurations according to our feature set. Through curve fitting, several models had an R squared of 1, including linear, suggesting that the sample of 150 and current number of known states is not enough to be able to estimate the maximum number of states. However, the linear model is able to suggest that for now, the approximate number of states will grow by 54.5 states every game that is run for self-learning.

1. *Real world Application*

The self-training program was timed using the UNIX time command, a single session of training, simulation and data storage took only 5.721 seconds. This overall time suggests an average of .09 seconds taken for the agent to take one turn. This speed per turn would be appropriate in an implementation that could be used in a game against a human player, and even in a speed backgammon game (10 seconds per move).

# *Discussion*

Due to our model not reaching a point where the growth of states is not slowing, additional training should be done to further train known states. This has potential implications to all of our results, and suggests that it is very likely the agent will have to select between untrained states. In TD-Gammon millions of iterations were trained, with our current linear model that suggests at least 54 million states, leaving our current number of states known (56 thousand) less than 1% of possible states. Currently non-distributed computing is being used for training, which at the current rates would take over 65 days to reach one million iterations, so to continue to match the depth of TD-Gammon, high performance computing should be utilized in order to train at a practical rate.

References

1. Tesauro, Gerald. “Practical Issues in Temporal Difference Learning.” Reinfocement Learning, vol. 8, 1992, pp. 33-53., doi:10.1007/BF00992697
2. G. Tesauro, “Neurogammon: a neural-network backgammon program,” 1990 IJCNN International Joint Conference on Neural Networks, San Diego, CA, USA, 1990, pp.33-39 vol.3.
3. Wang, S., Yu, M., Guo, X., Wang, Z., Klinger, T., Zhang, W., Chang, S., Tesauro, G., Zhou, B., & Jiang, J. (2018). R3: Reinforced Ranker-Reader for Open-Domain Question Answering. AAAI.
4. Tesauro, Gerald. “Programming Backgammon Using Self-Teaching Neural Nets.” *Artificial Intelligence*, vol. 134, no. 1-2, Jan. 2002, pp. 181–199., doi:10.1016/s0004-3702(01)00110-2.
5. Tesauro, G. Simple neural models of classical conditioning. Biol. Cybern. 55, 187–200 (1986). https://doi.org/10.1007/BF00341933